Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Computer Vision Analysis of the body-pose similarity from two different subjects with the aim of the correct development of physical exercises.

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Sergi Macián Ribera

ADVISOR: Francesc Tarrés Ruiz

DIRECTOR: Jordi Jiménez del Hierro

DATE: October 23rd, 2020

Title: Computer Vision Analysis of the body-pose similarity from two different subjects with the aim of the correct development of physical exercises.

Author: Sergi Macián Ribera

Advisor: Francesc Tarrés Ruiz

Director: Jordi Jiménez del Hierro

Date: October 23rd, 2020

Abstract

The aim of this project is to obtain a new Neural Network model capable to properly detect specific keypoints on human bodies. These keypoints will be later treated for real-time corrections in the field of sports and rehabilitation exercises.

Generally, keypoint detection models focus on unconstrained environments; training and testing images contain one or more people, they might be practicing different activities, people may not be centred in the image, different clothing and background, etc. However, this project has focused on a more constrained context. There is a specific activity that the main subject is practicing; sports. And, moreover, only one person appears in the middle of the image and there are not object occlusions. In order to train the model, we have performed a fine-tuning on an open-source model from PyTorch with an open-source dataset that focuses on sports; LSP. We have then analysed if by constraining the context, the neural network model performance is improved.

The conclusion that we have reached is that LSP dataset is not correlated enough with real case scenarios in which a person is practicing sports in front of a camera. The model we have trained is capable to estimate keypoints on LSP images with high accuracy but, despite of that, when the model is used in a real case scenario, model predictions have not been as good as expected.

Acknowledgement

I would like to express my gratitude to CIMNE, the company that gave me the opportunity to develop this project. Specifically, Alberto Burgos, who always gave me a hand when I was lost and Jordi Jiménez and Ángel Priegue who were always understanding and supportive. Besides CIMNE team, I would like to thank my family for the unconditional support and patience, especially during the coronavirus lockdown. They also recorded the videos that have been used in the evaluation of the project. Lastly, I would also like to thank Francesc Tarrés Ruiz for accepting this project and mentoring me, and to whom I am grateful for criticism, advice and suggestions.

CONTENTS

INTRODUCTION

The aim of this project is to develop a tool that helps people to properly perform physical exercises at home. With on-edge Artificial Intelligence (AI) technologies it is possible to check and correct user's performance in real-time. By using Deep Learning algorithms on devices, we can get the pose of a subject, compare it with the pose of a professional trainer and correct the person in real-time.

Previous to the correction, a professional trainer has to submit a video of himself or herself performing the exercise. This video will be processed in a server with a high-accuracy keypoint detector model, i.e., this model focuses on reliability instead of speed. Once the keypoints are obtained they will create sets of "rules" that later on will be checked when the person is executing the exercise.

In this project we have focused on training a model that must be capable to extract keypoints from the trainer video with high accuracy. It corresponds to the phase 1 detailed in the following Fig. 0.1 in orange colour.

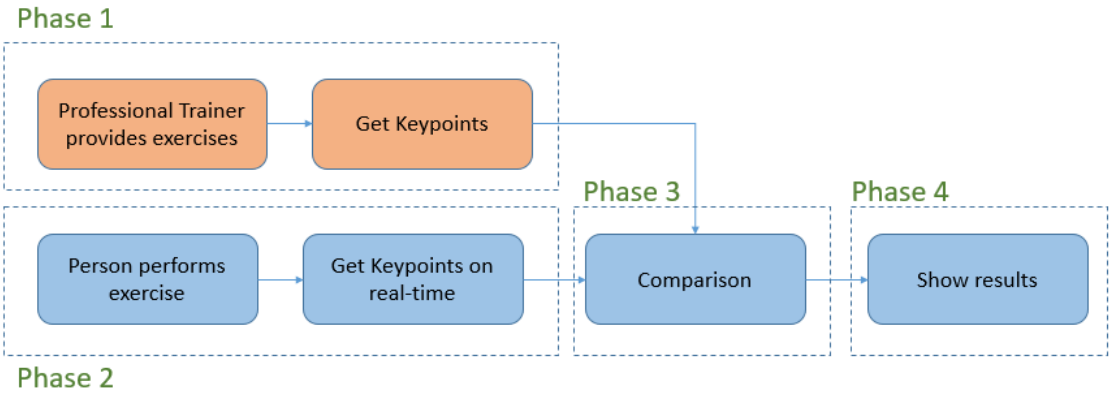


Fig. 0.1. Phases of the whole project and, in orange, the specific study that we have explained in the current document.

We have first had a look at the state of the art. We wanted to know if there were open-source models that were already implementing the functionalities we require. We have concluded that there are several outstanding models already trained but they are not open-source or they have commercial limitations. As we would like to build a product from this study, we have had to discard a lot of these models. The same has happened with datasets.

After our study of the art analysis, we have decided to perform a fine-tuning on a pre-trained model from PyTorch with an open dataset: LSP. Fine-tuning is a Transfer Learning technique that consists on training a previously trained model in order to improve its performance or to adapt it to the users' needs. Training a network from scratch requires high resources and time expenses. Therefore, we can save a lot of time and resources by using this technique.

Keypoint detection is normally studied under unconstrained environments, i.e., there are one or more people in every image, people are practicing different activities, they have different clothes, backgrounds, etc. We have tried to improve our predictions by constraining the context; a single person centred in the middle of the image that is practicing sports and has no object occlusions. Then, we have decided to use LSP dataset because it is an open-source dataset that accomplishes all these mentioned requirements. The dataset contains a total of 2000 images and every image is annotated with 14 different keypoints that build the human body structure.

The model we have chosen for the fine-tuning is an open-source pre-trained model from PyTorch. This model is capable to detect a box that contains a human body and then, find the keypoints inside of it. We have analysed it and we have noticed that its accuracy is not good enough for our purpose. Our objective is to improve this models' performance by running a fine-tuning process on it.

After performing some fine-tuning trainings, we have obtained a model that is capable to properly predict keypoints on LSP evaluation images. However, we have noted that these fine-tuned model, even if it is good for LSP detection, it is not good enough for some real case scenarios. In order to prove so, we have recorded several videos that contain a single person practicing exercises in the middle of the scene, just like LSP does. However, when the model is run on these videos, the returned inference is not as good as we desired. Some mistakes like keypoint swapping and keypoint missing do still occur. Then, we have concluded that LSP dataset is not correlated enough with our real case situations and, therefore, we should use a different dataset.

In this Master Thesis document, we will first present some theoretical background related to Machine Learning applied on Images in chapter 1. Then, in chapter 2, we will see more specific theory and State of the Art related to our challenge; Person Pose Estimation. After that, we describe the methodology we have followed in order to fine-tune the model in chapter 3. And, finally, the obtained results are presented in chapter 4.

CHAPTER 1. MACHINE LEARNING ON IMAGES

Machine Learning is a field of Artificial Intelligence that allows algorithms to evolve and take decisions like if they were human decisions. Algorithms can be trained in order to learn what results they should return according to a given input (Deep Learning) or to learn how to act according to environment events (Reinforcement Learning). This project focuses on Deep Learning branch since the objective is to end up with a model that is capable to return accurate keypoints locations depending on an input image.

Machine Learning is not a new topic, back on the 60's there were already studies that were working on this field. However, since the introduction of powerful GPUs that can process in parallel the huge amounts of computations that must be calculated on ML algorithms, this field has become popular and accessible for everyone.

1.1. Deep Learning

Deep Learning is essentially a set of non-linear transformations that are applied to an input variable in order to represent the information in the most natural way. These transformations facilitate the input's recognition and classification. Then, we can create an architecture that contains all these non-linear transformations together with huge amounts of weights which can be tuned in a training process in order to generalise what information should the model return according to the input variable.

Deep Learning can be applied to several fields such as image, speech, text, etc. In this particular project we have focused on image processing, instead of using regular Neural Networks, we have used Convolutional Neural Networks.

There are several image related challenges that are nowadays under study. These main challenges are the classification problem, which tries to identify and label objects; segmentation, that tries to outline objects and classify them, and, finally, keypoints detection or pose detection, that tries to identify some specific points on human bodies. In this project, we have focused on this last challenge.

In this current project we have used a supervised algorithm. I.e., we will fit data into the model and tell it what is the desired output. The model will try to adapt itself in order to be able to identify similar outputs for similar inputs. In unsupervised algorithms, no desired output is given as an input in the training and the model tries to classify the outputs itself.

1.2. Neural Networks

An artificial neural network gets its name due to the relationship with human brain neural networks, where neurons are the base element. As a neuron, you will decide what information you send depending on the information you have received. The combination of all neurons build a network, also known as architecture that, in AI field, is human-designed. This network can be modified and trained from trial-error techniques. Artificial neural networks can be compared with human neural networks; we often learn from our decision mistakes and so does an artificial neural network.

Neural networks are powerful because, optimally, they can predict a result, an output, generalising from what they have learnt from the training data. I.e., they do not learn by heart what output they must give according to a specific input. The main elements of an artificial network are artificial neurons, or perceptrons, which structure is depicted in Fig. 1.1.

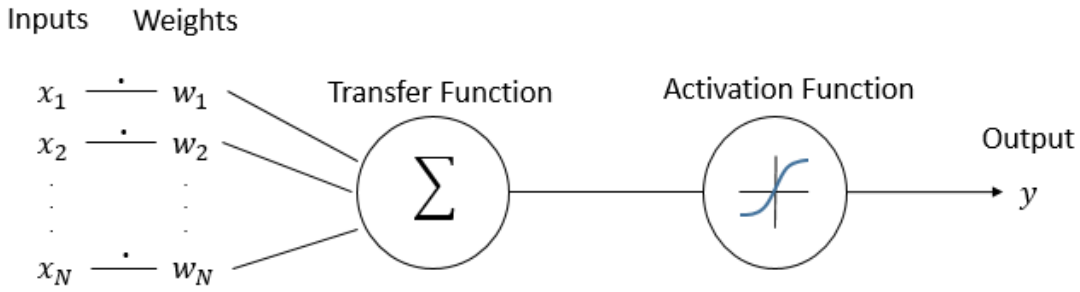


Fig. 1.1. Insights of an Artificial Neuron, also known as perceptron.

As we can see, a perceptron has several inputs that will be multiplied by some weights. These weights determine how important inputs are. All the inputs and weights are summed up and, depending on the result, the activation function will decide what output the neuron is transmitting to the next neuron.

The activation function of the perceptron is a mathematical equation that determines an output according to the combination of inputs and weights that are received by the perceptron. Its goal is to introduce non-linearities into the network since the majority of data that we find in the real world behaves in a non-linear way. Therefore, the algorithm can learn from the non-linearities that come from the data. If a neural network is built just with linear functions, even if there are a lot, the output will still have a linear relationship with the input.

There are several activation functions that can be used. One of them is ReLU which is commonly used because it lets the network converge quickly and because it is non-linear. Another common activation function is Softmax since it allows classification. It gets an input vector of size K and returns a vector of the same size that indicates the probability of each vector element. The higher the value, the more probabilities of being the correct class. Because of that, it is normally used as the output layer of classification networks. Their equations are

presented in 1.1 and 1.2, respectively. Other common activation functions are Sigmoid, binary step function and TanH.

$$F(x) = \max(0, x) \tag{1.1}$$

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for \; i = 1, \dots, K \; and \; z = (z_1, \dots, z_k) \in R^k \tag{1.2}$$

Overfitting is also a common issue that must be taken into account when training a neural network. We have an overfitted model when it is not able to generalise. We can recognise that a system has been overfitted by seeing that the error in the training dataset has been reduced while the error in the validation dataset has increased. It can occur in two situations, either the training input dataset does not represent the whole dataset (training and validation) or that the amount of parameters in the network is big enough to get adapted to the training data instead of learning how to generalise from it. We can see an example of this situation in Fig. 1.2.

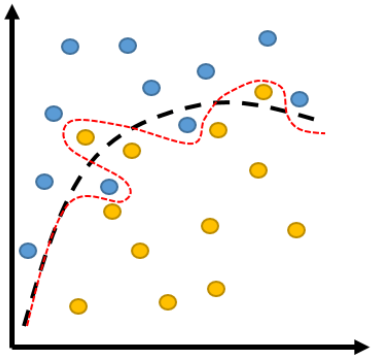


Fig. 1.2. Overfitting example. In black, the generalised model (correct). In red, the overfitted model (incorrect).

1.2.1. Convolutional Neural Networks

Convolutional neural networks (CNN) are a type of deep neural networks that are mostly used in image applications. They use a kernel, also known as filter or feature detector, that moves along the input image, conceived as a matrix, creating a new matrix, named activation map or feature map. Such map contains features and information about the input image. Kernel values are considered to be the weights of the network and, therefore, they are updated in every backpropagation process. As images are normally defined by RGB colours, we can define a 3D kernel that takes values from the three layers or a 2D kernel that just takes into consideration one of the three colours.

Before we work with CNN, we must also know the meaning of the parameters stride and zero-padding. Stride defines the jumping steps of the kernel over the image. Zero-padding defines the amount of columns and rows of zeros that we have to locate around the image. We add zero padding rows to avoid losing information from the edges of the image. Stride and padding define the size of the feature map according to the following equations.

$$w_{out}, h_{out} = \left(\left\lfloor \frac{w_{in} - K_w + 2p}{s_x} \right\rfloor + 1, \left\lfloor \frac{h_{in} - K_h + 2p}{s_h} \right\rfloor + 1\right) \quad (1.3)$$

Where w_{out}, h_{out} stand for output width and output height of the feature map, w_{in}, h_{in} stand for input width and height of the image, K_w, K_h stand for kernel width and height, p is the number of rows or columns added in each side and s_x, s_y refer to the stride value in x and y. If we consider a squared input image, a squared kernel and the same stride in horizontal and vertical, we can simplify equation 1.3 and obtain 1.4.

$$w_{out} = h_{out} = \left\lfloor \frac{N - K + 2p}{s} \right\rfloor + 1 \quad (1.4)$$

Where N is the image size, K is the filter size and s is the stride. There are multiple configurations of padding, kernel and stride values and we should consider which is the optimal one for our study case. In Fig. 1.3 we can see a basic example of a convolution with $N = 3, s = 1, p = 1$ and $K = 3$.

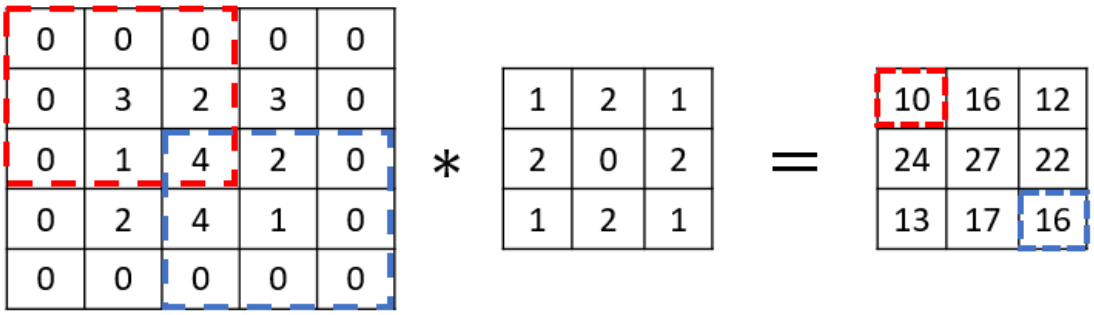


Fig. 1.3. Example of Image and Kernel convolution.

A common term that we also find related to the CNN topic is “channel”. By channel we refer to a layer’s depth. A RGB image, for instance, has three channels, Red, Green and Blue. When we create a Convolutional Layer, we can fix as many parallel layers as we want, each layer will come from a different kernel and will retain different information. For example, we can create a horizontal edge detector kernel and in parallel, a vertical edge detector kernel. We can see an illustrative example in the following Fig. 1.4.

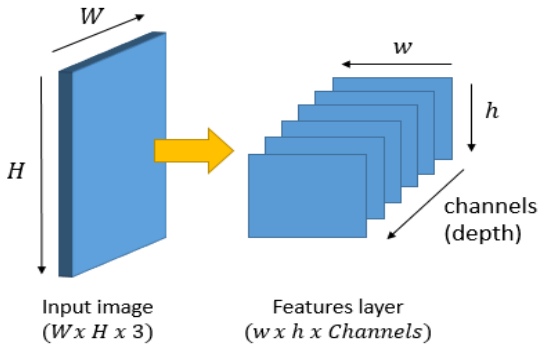


Fig. 1.4. We can obtain several parallel channels from a single image.

The convolution map also includes an activation layer right after the computation of the convolution in order to introduce some non-linearities. I.e., it works like the perceptron shown in Fig. 1.1 but instead of having an input vector, we have an input matrix.

In CNN architectures, it is common to find pooling layers. These layers reduce the amount of neurons, selecting the most important ones. A maximum pooling is normally used as the pooling layer and its objective is to select the maximum value from a set of values. We must also define a stride and kernel size value. We have represented its working procedure in Fig. 1.5.

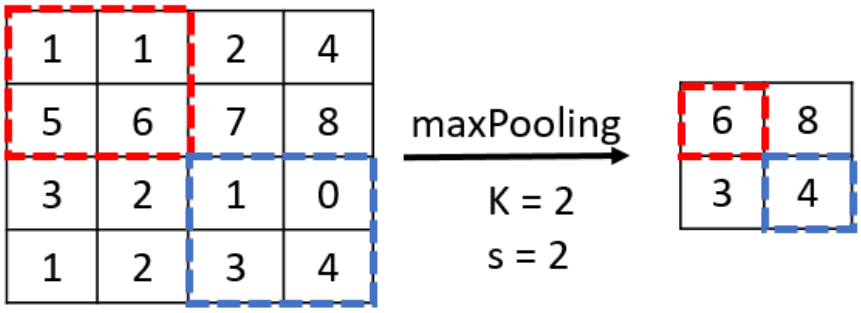


Fig. 1.5. Max Pooling. The maximum value from each 2x2 matrix is taken.

We can compute the size of the output matrix from a maxPooling applying the following equations 1.5, 1.6 and 1.7 where the initial matrix has dimensions $W_1 \; x \; H_1 \; x \; D_1$ and the output matrix has dimensions $W_2 \; x \; H_2 \; x \; D_2$.

$$W_2 = \frac{W_1 - K_x}{s_x} + 1 \tag{1.5}$$

$$H_2 = \frac{H_1 - K_y}{s_y} + 1 \tag{1.6}$$

$$D_2 = D_1 \tag{1.7}$$

1.3. Architectures

Neurons can be grouped by layers. Neurons that are connected to the input value or values, perform the input layer. Neurons that give an output result build the output layer. And, in between, there are the hidden layers. All these layers together build the whole neural network, known as architecture.

There are different ways to connect these layers depending on our project goals. E.g., we can connect an input layer, a couple of hidden layers and an output layer, as depicted in Fig. 1.6. This creates a basic architecture known as MLP, Multilayer Perceptron. In this case, as all neurons from a layer are connected to all neurons from the following one, this network can be called Fully Connected (FC) network.

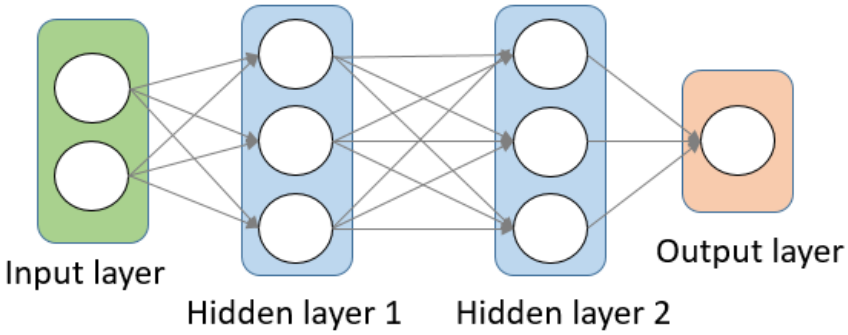


Fig. 1.6. MLP architecture example.

In this current project we work with images as inputs so we will work with CNNs. This kind of architectures normally combine a feature extractor built with convolutional layers and max poolings and, at the end, some fully connected layers to get the final conclusions about the obtained features. The flatten process consists on distributing all information contained in a 3D matrix into a vector. An example of this scenario is depicted in Fig. 1.7.

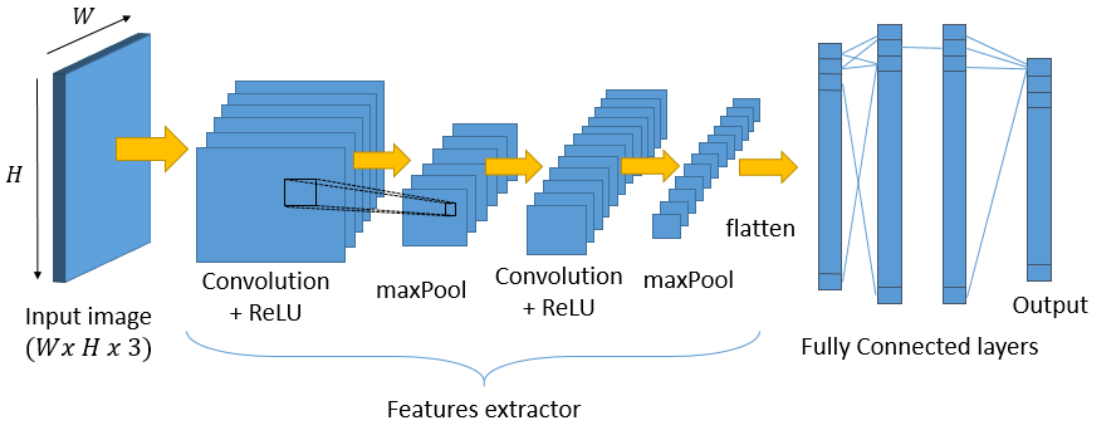


Fig. 1.7. Example of convolutional neural layers and fully connected layers in the same architecture.

In some fields, like image classification, the feature extractor is also known as the backbone of the architecture. There are several backbone designs that have been proposed in different studies for image classification and, as we will later see, they are also useful for keypoint detection. Some backbones contain more layers than others and, therefore, there is a trade-off between speed and accuracy. Bigger backbones with more layers tend to be more accurate, but also require more processing time, so they are more slow. In [1] we can see a speed-accuracy trade-off study about several backbones used for object classification.

In object detection field several algorithms have been studied in order to obtain and classify objects in an image. R-CNN [2] and Fast R-CNN [3] are two algorithms that could solve object detection challenges, but they were too slow to be used in real-time. Then Faster R-CNN [4] was released and became the new

benchmark. It introduced the usage of Regional Proposal Networks (RPN) and it got better than the previous time-consuming algorithms.

Architectures that deal with R-CNN (Regions based CNN), important for object detection and for keypoint detection, can implement a feature extractor named FPN (Feature Pyramid Network) that works in parallel with the backbone. Its objective is to obtain information from all the feature layers, not just from the last one. First feature layers, closer to the image, contain more spatial information; the last feature layers, further from the image, contain more semantic information. Thanks to this parallel backbone the model ends up learning from both spatial and semantical information. Because of that, this method is good at detecting and extracting features of objects of different sizes. Moreover, it barely adds computing time. In Fig. 1.8 we can see a summarised schema of FPN.

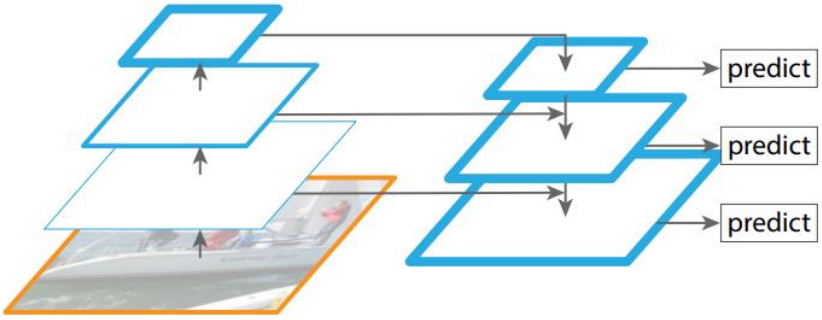


Fig. 1.8. Feature Pyramid Network (FPN) schema. Fig 1.(d) in [5].

Region Proposal Networks (RPN) is a kind of network that is capable to propose regions, called anchors, from feature maps that have been obtained from a CNN. The proposed regions and the feature maps go through a RoI pooling layer. This layer acts like a Max Pooling layer but with a variable input size. It works as follows: it crops the feature map according to the proposed region coordinates, it divides the map into smaller matrices and it gets the maximum of each matrix as is represented in Fig. 1.9.

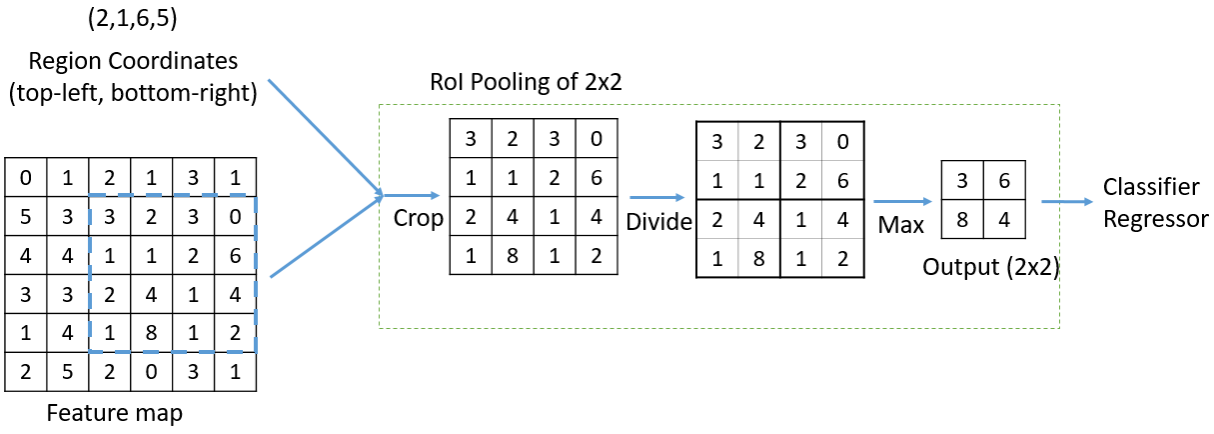


Fig. 1.9. RoI pooling of 2x2 example.

RoI pooling layer output is sent to the object classifier and regressor networks that will obtain the final output for each one of the proposed regions. An illustrated scheme is shown in Fig. 1.10.

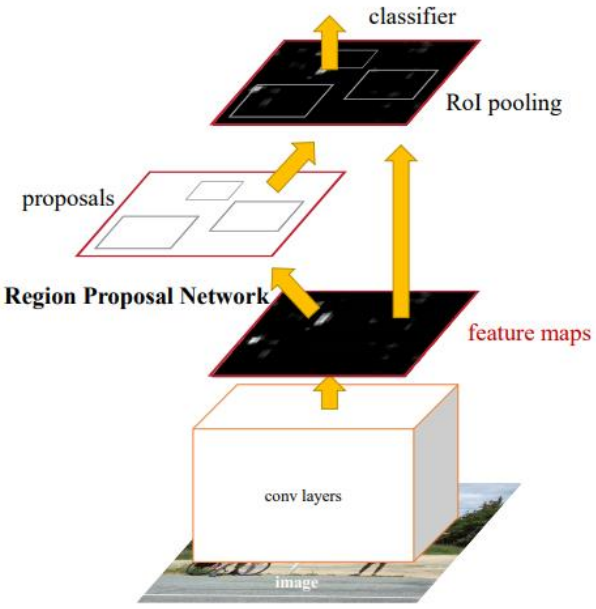


Fig. 1.10. Faster R-CNN schema. Figure 2 in [4].

Object detection and keypoint detection are close related fields since they both detect elements in an image and then, classify them. In our project we have used a base model named keypointRcnn_ResNet50_fpn from torchvision [6] that implements the following architecture depicted in Fig. 1.11.

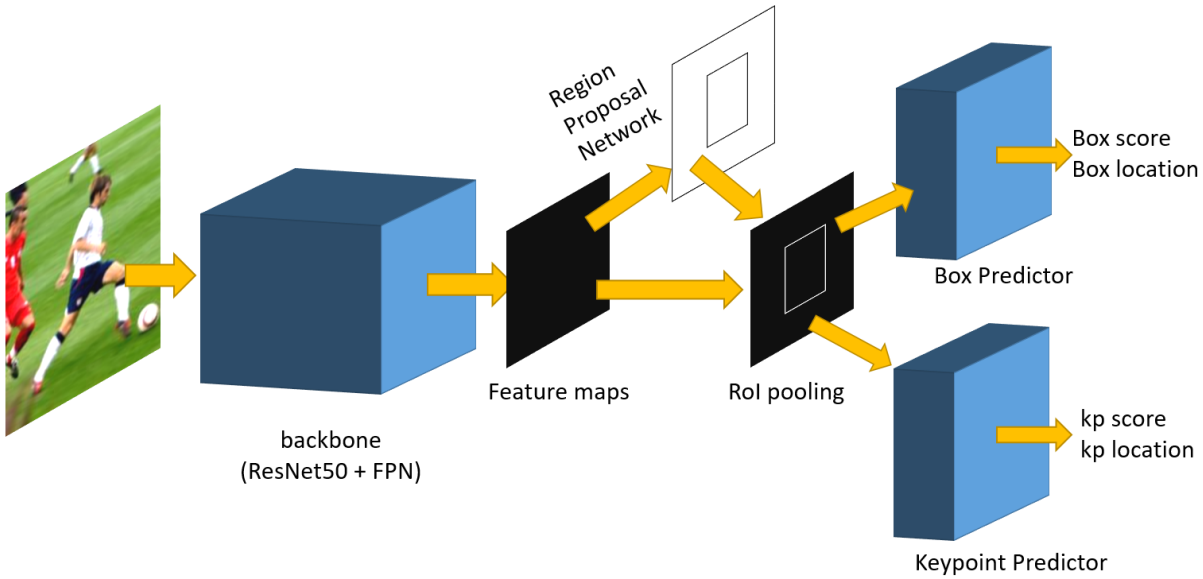


Fig. 1.11. KeypointRcnn_ResNet50_fpn model architecture that has been used in the current project.

1.4. Training

The process of training a model in supervised environments means that several inputs, together with their desired output values, called ground truth values, are given to a neural network architecture. The model processes the input and ends up with an output value that is compared with the desired one, obtaining an error value, for example, from a MSE function. This value is also known as the cost function and the model tries to learn how to minimize it. In order to do so, it computes the gradient of the cost function for all the unfrozen weights of the network. This gradient points out the optimal direction in which the cost function will be minimized.

Normally a model is trained for several iterations over the whole training dataset, called epochs. After every epoch, the model can be tested with the validation dataset. In such validation, an evaluation function is computed with the obtained output and compared with the ground truth. It returns a value that indicates how well the model is performing in validation data, which has not been used in the training. If this metric improves over the previous epoch, we can save the new trained model. There's a moment in which this metric stops improving over time or that it even gets worse because of overfitting. At that point, we can stop the training. Training dataset and validation dataset must be different; they cannot repeat images among them.

1.4.1. Optimizers

There are several ways to update weight values. Optimizers are the algorithms that decide how and how often the architecture weights are updated. Optimizers also have control over the learning rate and they can even set different learning rate values for different layers of the architecture. Depending on the chosen optimizer, we may end up with a better or worse model, i.e., a model with a lower or higher loss value.

Weights can be updated after seeing the entire training dataset with a Gradient Descent (GD) algorithm. The first order derivative of the loss function is computed and backpropagated along the network. Its drawback is that it requires a lot of memory to store and calculate the gradient of the whole dataset. Its equation is defined as following:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \nabla F(\boldsymbol{w}_t) \tag{1.8}$$

Where $\boldsymbol{w}_{t+1}$ is a vector of the new obtained weights, $\boldsymbol{w}_t$ is a vector of the current weights, α is the learning rate and $\nabla J(\boldsymbol{w}_t)$ is the gradient of the loss function computed with the current set of weights. The gradient points out the direction of the minimum.

In order to avoid retaining information about all the errors until the whole training dataset has been forwarded, we can train the network with a single image or with small set of images, called batches. In these situations, we are referring to

Stochastic Gradient Descent (SGD) and Mini-batch SGD. This kind of training is noisier, i.e., the loss value does not always face the minimal point direction, but it still converges faster than GD algorithm and it does not require a large computing memory. The equation is the same one as the one for GD in 1.8 but instead of computing the error cost function $F(\boldsymbol{w}_t)$ for the whole dataset, it just uses a small amount of samples.

In order to decrease the noise from SGD and soften the convergence, we can use momentum hyperparameter which takes into consideration previous weights values in order to update the new ones. The SGD + Momentum algorithm can be described as follows in equations 1.9 and 1.10 as written in [7].

$$\boldsymbol{v}_{t+1} = \mu * \boldsymbol{v}_t + \nabla F(\boldsymbol{w}_t) \tag{1.9}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha * \boldsymbol{v}_{t+1} \tag{1.10}$$

Where $\boldsymbol{v}_{t+1}$ stands for velocity vector in the next iteration, $\boldsymbol{v}_t$ is the velocity vector in the current iteration and μ stands for momentum hyperparameter.

There are more algorithms of neural networks optimizers like Nesterov Accelerated Gradient (NAG), Adam, Adagrad and Adadelta. In this current project we have decided to use Mini-Batch Gradient Descent with momentum algorithm. Every batch contains 2 images and momentum value has been set to 0.9. This algorithm is already implemented in a PyTorch function and it can be found in the library [7]. Its name is SGD but it actually runs as a Mini-Batch GD because we input a batch of 2 images instead of a single one.

1.4.2. Learning Rate

Learning rate is an important hyperparameter that has appeared in the previous equations 1.8 and 1.10. Its main function is to control how much weights learn from the gradient. Its value has to be chosen carefully because if it's too small, the network may spend a lot of time to train and if it's too large, it may diverge, i.e., it may never find the local minima. An illustration is depicted in Fig. 1.12.

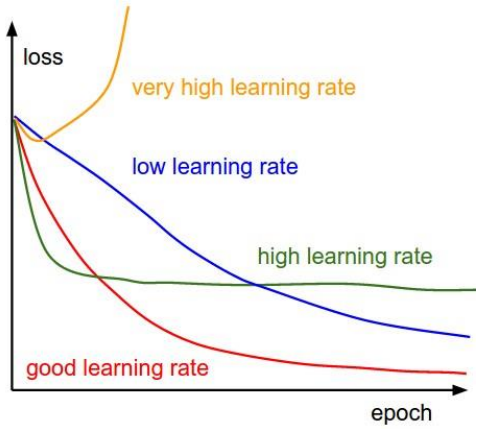


Fig. 1.12. Effects of different learning rates. Image obtained from [8, p. 231]

The optimal learning rate (LR) value can be found by an iterative process called “Learning Rate Finder” method [9]. We set a very small LR value, train the network for one iteration with a single image and register the output loss. Then, we increase a little bit the LR value, train again the network for one iteration and register the new output loss. We repeat this process until we obtain the LR value that is large enough to produce divergence, i.e., that the loss starts increasing exponentially. If we plot the LR values, in a logarithmic scale, together with the registered losses, we obtain a graph like the one depicted in Fig. 1.13.

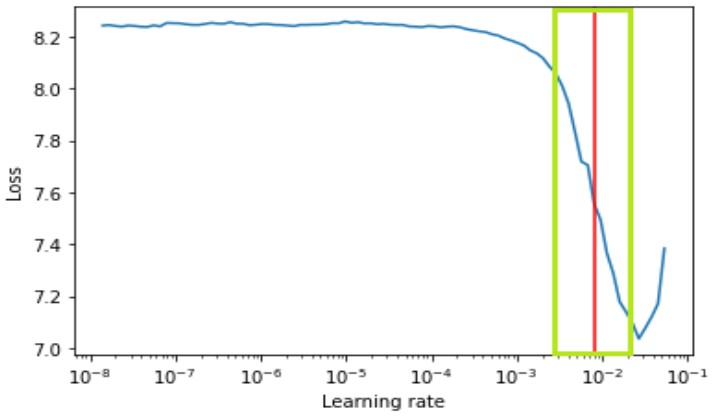


Fig. 1.13. LR study. In green, the area in which the network will be properly trained according to the LR values. In red, a possible optimal value.

In Fig. 1.13 we can see at the left side of the graph the loss is not getting reduced since the learning rate is too low. At some point, it starts decreasing until a certain point in which it starts increasing again. We must choose a value that is contained in the descending slope, where the loss decreases the most.

It happens that after some training epochs, the loss function does not decrease anymore. The model may be close to the minimum value but as its taking too big steps, i.e., too big learning rate, it’s bouncing from one side to the other of the cost function. In such situation, reducing the learning rate can be useful. We can see an example of why this happens in Fig. 1.14.

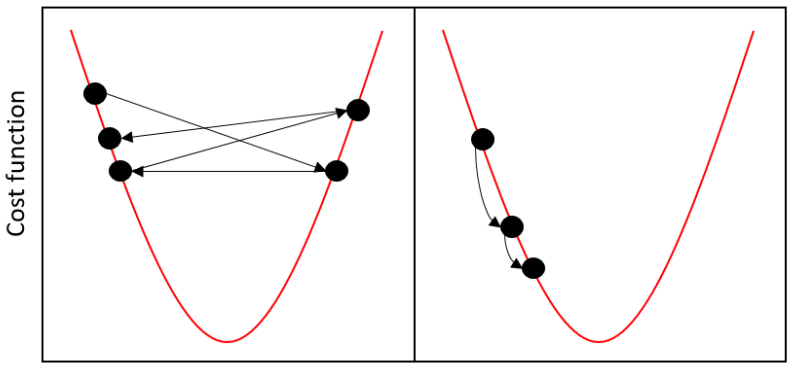


Fig. 1.14. (Left) Loss results after training when LR is too high. (Right) Loss results after training when LR has been reduced. As the step is smaller, they can get closer to the local minimum.

There are several techniques to reduce LR values during the training. In the upcoming sections, we will present StepLR and Plateau algorithms. We will also introduce CLR algorithm that decreases and increases the learning rate cyclically.

1.4.2.1. Step LR

It consists on reducing the learning rate after a specific number of iterations that can be set by the user. For example, we can decide that after a specific amount of batches or epochs, the learning rate gets reduced by a fraction that can be also set by the user.

1.4.2.2. Plateau

Plateau algorithm checks the value of a function that we want to minimize, like the loss function, or that we want to maximize, like the evaluation function, at the end of every epoch. If the function has not been minimized or maximized after a specific number of epochs, called patience, the LR gets reduced by a fraction. Both the reduction fraction and the patience values are chosen by the user.

1.4.2.3. CLR

Cyclical Learning Rate (CLR) is a type of learning rate explained in [9]. Instead of using a single value for the learning rate and decrease it over time, the study suggests a learning rate that varies cyclically within a range of values. In the research it is said that SGD and CLR work properly together because they reduce drastically the number of iterations needed to reach the best accuracy.

In order to use CLR method we have to define a maximum bound, a minimum bound and a stepsize, as shown in Fig. 1.15. According to [9], Stepsize should be 2 to 10 times the number of iterations in an epoch. E.g., if we have 50.000 training images and the batch size is 100, the total amount of iterations is 50.000/100 = 500 iterations and, therefore, the stepsize value can be from 100 (500*2) to 5.000 (500*100). The maximum bound and the minimum bound can be set from Fig. 1.13. The smaller value in the green area indicates the minimum bound and the higher value in the green area indicates the maximum bound.

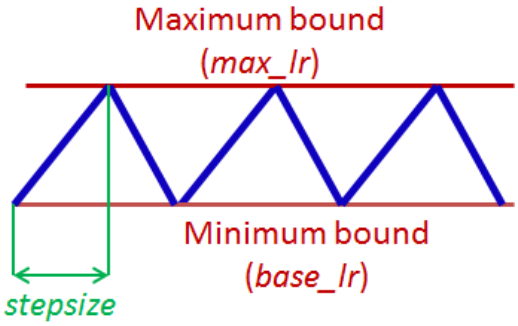


Fig. 1.15. CLR configuration. Figure 2 in [9].

CHAPTER 2. POSE ESTIMATION PROBLEM

In this computer vision project, the terms “pose estimation”, “keypoints” and “joints” appear several times. They all refer to a set of specific points located on a human body.

Keypoints may refer to face-keypoints, body-keypoints or even hand-keypoints. In this project we are focused on sports and, therefore, we have used body-keypoints. There is not a single way to define these body-keypoints as we will later explain in section 2.1. Each dataset defines them in its own manner.

With the latest improvements of Neural Networks, in particular Convolutional Neural Networks (CNN), it has been proved that this kind of technology works properly when trying to solve pose-estimation problems. There are some studies and projects that work with 3D images; they locate a huge set of keypoints in order to build a 3D model of the person. This technique is useful for virtual reality purposes. There are some 3D cameras available in the market that are able to obtain the 3D keypoints such as Kinect, from Microsoft Team. However, what the majority of people have at home is a mobile phone device or a laptop with a 2D camera. Hence, our project has tried to detect the pose of a single person from a 2D frame, obtained from a 2D video, making the technology accessible to everyone.

The most common errors that we have to face when working with keypoint detectors are the following ones: jitter, inversion, swap and miss. Jitter refers to a small location error; the keypoint is detected around the area where it has to be, but it’s not properly centred. Inversion happens when keypoints from the right side are assigned to the left side or vice versa. E.g., the left knee is estimated to be in the position of the right knee. Swap is an issue that occurs when the keypoints of two different people are exchanged. And, finally, miss happens when a keypoint is not detected even if it’s visible.

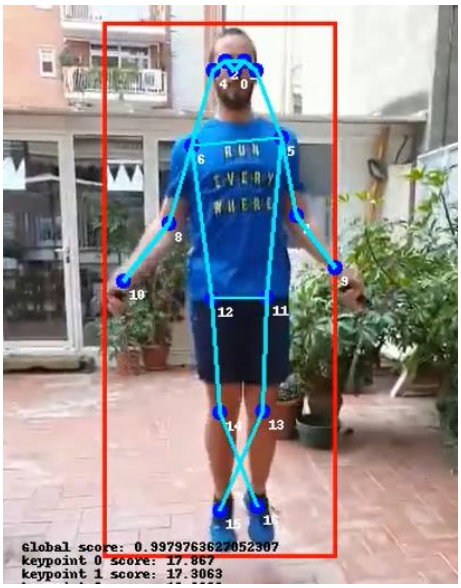


Fig. 2.1. Swap example. Right and left ankles are swapped.

Keypoint detection tries to be as generalist as possible, i.e., it tries to predict peoples' position in unconstrained environments and, therefore, the training of the model does not consider a specific action, background, clothing nor anatomy. Self-occlusions, object occlusions and different human body orientations are also common issues that affect the performance of the model.

2.1. Keypoints datasets and topologies

In this section we will explain the different topologies of body-keypoints; different ways to locate keypoints on a person. Nowadays there's no standard for such task. Every dataset defines keypoints' topology in a different manner as we will see now.

2.1.1. COCO Dataset

COCO is a large open dataset. It contains more than 330.000 images and more than 200.000 of them are labelled. This dataset provides tagged images for features like Object Classification, Object Segmentation and Keypoint Detection. It contains more than 250.000 people tagged in the images. We can find more information in the paper [10] and the official website [11].

Every year a competition takes places. Researchers can submit models for the before mentioned features such as Object Classification. Then, they are evaluated and COCO publishes the results in the official website. From there, a list of the best models can be seen. Some of them have links to their GitHub repository or university website.

In terms of keypoint detection, COCO uses the 17 keypoints represented in Fig. 2.2. When working with keypoints, it focuses on 58.945 with 156.165 tagged people with a total of 1.710.498 keypoints as it's written in [12].

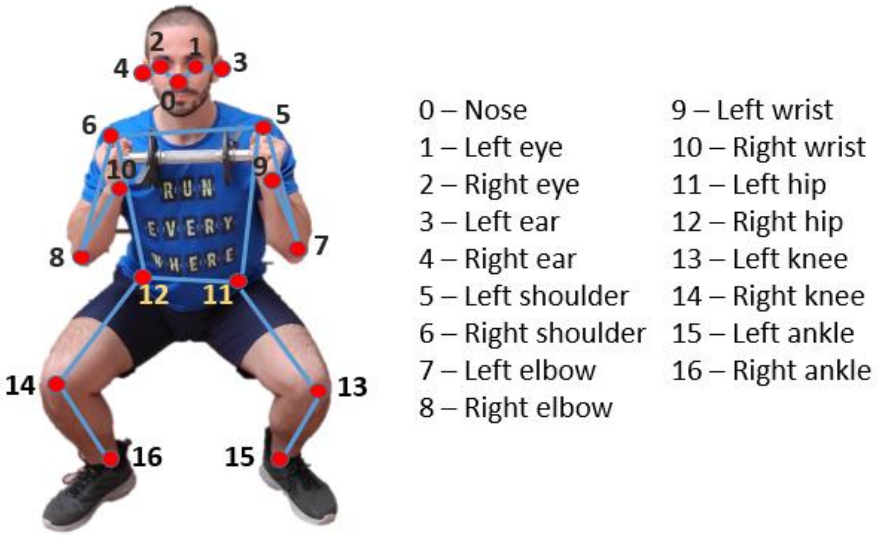


Fig. 2.2. Keypoints' locations on COCO Dataset.

COCO ground truth keypoints are located with x and y coordinates representing pixels. The (0,0) point corresponds to the left top corner. Moreover, COCO keypoints also contain visibility information [12]. Visibility equal to 0 stands for “not labelled”; the keypoint is outside of the image. Visibility equal to 1 means that the keypoint has been labelled but it’s not visible and it’s equal to 2 when the keypoint is labelled and visible.

According to FollowMeUp Sports [13], around 85% of the images in Coco Dataset contain subjects on standing postures. This may affect negatively when trying to train a model to detect irregular postures, like yoga or sports poses.

2.1.2. LSP Dataset

Leeds Sports Pose (LSP) Dataset contains 2000 images with a single person of roughly 150px in length per image practicing different kinds of sports. Unlike COOC dataset, in LSP dataset every person has 14 keypoints labelled, supressing the ones located on the face. Its usage is open and free, even for companies. LSP images were obtained from Flickr platform. The keypoints distribution can be seen in the following Fig. 2.3 and in Table 2.1. More information about this dataset can be found in [14].

As it happens in COCO and many other datasets, keypoints are composed of x and y coordinates starting from top-left corner and a visibility tag. After doing a research on this Dataset we have concluded that the visibility values are defined as follow:

- V = 0 labelled and visible
- V = 1 labelled but not visible

Fig. 2.3. LSP image (non-visible keypoints in red, visible keypoints in blue)

Table 2.1. Keypoints order in LSP Dataset.

0 – Right ankle	6 – Right wrist	12 – Neck
1 – Right knee	7 – Right elbow	13 – Head
2 – Right hip	8 – Right shoulder	
3 – Left hip	9 – Left shoulder	
4 – Left knee	10 – Left elbow	
5 – Left ankle	11 – Left wrist	

As people are practicing different sports, the orientation of the subjects changes a lot between images. E.g., there are images in which the person is standing and others in which the person is lying on the floor. There are several keypoints that are not visible due to self-occlusions but not from object occlusions, i.e., there are no objects in between the camera and the subject, but it can happen that a leg of the subject is occluding the other leg. Moreover, the subject is centred in the middle of the image. All these situations are similar to the ones we have faced in our project when a person wants to practice a sport in front of a camera.

2.1.3. LSP-Extended Dataset

LSP-Extended is the extended version of LSP Dataset. In this case, 10.000 images build the whole dataset. Images come from Flickr for the tags “parkour”, “gymnastics” and “athletics” and have been labelled by Amazon Mechanical Turk (AMT). It is said in the official webpage [15] that some images are not guaranteed to be highly accurate. We have analysed the dataset and we have found too many images that were not properly tagged. A couple of examples are shown in the following Fig. 2.4.

Fig. 2.4. LSP-Extended examples in which there are missing tags.

As we can see, there are some missing keypoints in both images. Therefore, we have decided not to use this dataset.

2.1.4. MPII

MPII is a dataset that contains around 25.000 images that contain more than 40.000 people with their annotated keypoints. Every image is labelled with a different activity and there are a total of 410 human activities. I.e., we could use images from certain sports like “bicycling”, “dancing” or even “bowling” from a total of 410 different activities. All the images have been obtained from YouTube videos. The dataset is open for non-commercial usage. Hence, we are not able to use it in our current project.

All annotated people are tagged with a total of 15 keypoints and they are distributed as shown in Fig. 2.5. More information can be found in the official website [16].

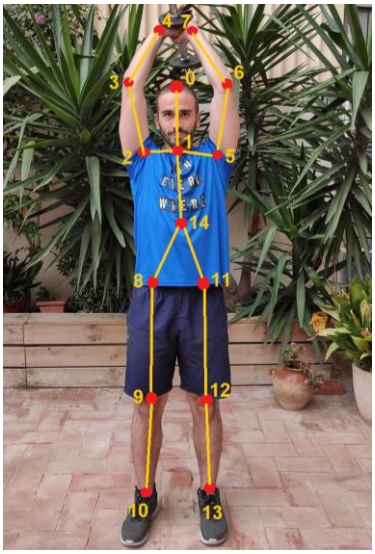


Fig. 2.5. MPII tagged keypoints.

2.1.5. Yoga-82

Yoga-82 is a recently created dataset that brings human joints recognition to a specific field, as its name indicates, it is based on Yoga positions. “Existing datasets for learning of poses are observed to be not challenging enough in terms of pose diversity, object occlusion and view points” [17]. This dataset could be useful for our project since we face similar challenges. However, it is not available for non-commercial research and, therefore, we are not able to use it.

2.1.6. PoseTrack

PoseTrack dataset helps researchers to face “Multi-Person Pose Estimation” as well as “Multi-pose Pose Tracking” problem. It provides a total of 1356 videos and every video frame is annotated with the corresponding keypoints. There are more than 46.000 video frames annotated with more than 276.000 people on them. However, we face the same problem that occurs with other datasets, the availability is not public for commercial usage, as it is stated in the official website.

2.1.7. FollowMeUp Sports

This dataset has been published by the end of 2019. Therefore, it is a recent dataset that tries to be the new benchmark in terms of human keypoint recognition challenges. It contains information about more than 200 workout activities and it tries to cover a wider set of people's rotations and irregular positions. For every activity type, three different shooting angles have been used.

All these dataset properties are very related to our project challenges and, therefore, it would be very convenient for us to use this dataset. However, the dataset paper is the only information that has been published about this dataset. All the test and train sets, within the keypoints annotations have not been published yet.

2.2.8. Summary

There are several datasets that focus on full body-keypoints but the majority of them are not suitable for our project. The main reasons are the limitation of non-commercial usage, the lack of subjects practicing sports and the availability of those datasets.

After analysing all the previous datasets and deciding which procedure we want to follow in this project, a fine-tuning of a pretrained model, we have decided to use LSP dataset since it contains enough annotated images, 2000, that seem to be strongly related to our project scopes. There is one single person per image that is practicing sports and our final product will be in a similar environment. If we wanted to detect several people at once or we wanted to train a model from scratch, we would have probably used COCO as the main dataset.

2.3. Common architectures

There are two common ways to design human pose detector neural architectures. They are called top-down and bottom-up. Their main difference is the order of detection. Top-down detect first the person box and then detect the keypoints inside such box. I.e., they first find the person and then its keypoints. On the other hand, bottom-up algorithms detect first the keypoints and then they correlate them building up the person. I.e., they first find the keypoints and then the persons.

Bottom-up keypoints make use of FCN like CPM or Hourglass like the one in Fig. 2.6. These network architectures generate a new image from an input image, i.e., they work as an encoder-decoder. The output image is a heatmap that works as a confidence map. Pixels that are more likely to be a specific keypoint will be marked with a different colour than pixels that are not likely to build a keypoint.

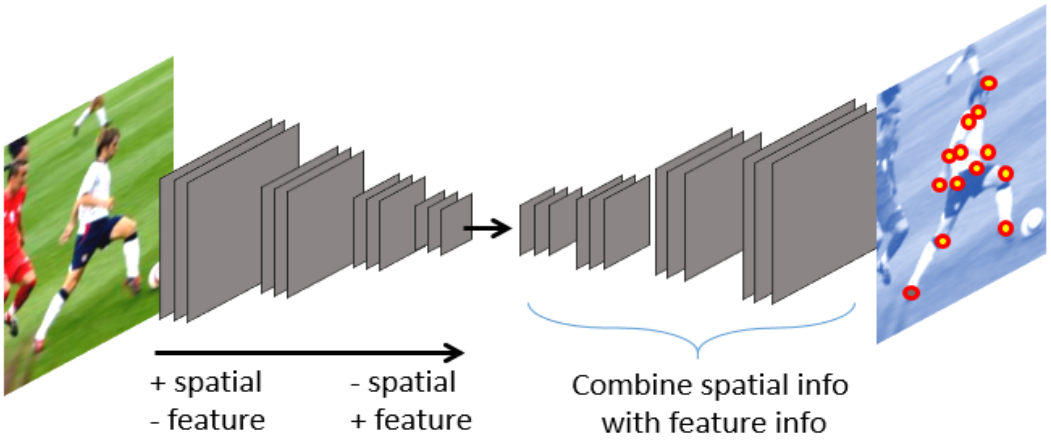


Fig. 2.6. Bottom-up keypoint detection example with Hourglass architecture.

When there are several people in an image, it can be difficult to link the detected keypoints in order to build up a whole human body. OpenPose project introduced a new technique that uses PAF (Part Affinity Fields) that help on the association of body parts with individuals. Then, apart from the keypoint confidence map, an affinity map is created. Then, the output image is given from the combination of these two heatmaps. OpenPose won the COCO 2016 Challenge by introducing this technique.

Top-down techniques use a CNN to produce a feature map that will be processed by a RPN which will detect persons. The proposed regions will then contain information about a single subject and will be analysed separately detecting the different keypoints. This kind of architecture is the one that we have used in the current project and it is explained with more detail at the end of section 1.3. An example of this architecture is represented in Fig. 1.11.

In the research [18] the pros and cons of both kinds of architectures are detailed. It is said that bottom-up architectures are good for occlusion and complex poses but are weak when referring to structural information, i.e., they have difficulties to associate body parts with individuals and, therefore, they create a lot of false positives. On the other hand, top-down architectures are strong against structural information, i.e., they are good on predicting human boxes, but they are not good enough for complex poses or crowded images. Research [18] suggests combining both architectures. Bottom-up for keypoint detection and top-down for reject false positives.

2.4. Pre-trained models

Nowadays, human pose detection problem has been studied by many universities and companies. Therefore, there are already some models that can estimate human keypoints. However, as we will see now, the majority of them are not available for everyone. Some have been developed by private companies and require a payment and others are just open for academic or non-profit organizations and, as we are developing a private project, we can't use these non-commercial solutions.

2.4.1. Non-commercial projects

The majority of universities or research centres want to share their experiments with other researches. However, they do not let companies to take advantage of their findings. That's why, they publish their results and models in an "open-source" way but with a license that states that its usage is limited for non-commercial activities.

OpenPose is a keypoint detection project that has been developed by a team of researchers that work together with CMU Panoptic Studio Dataset. They have created several 2-D models capable to detect up to 135 keypoints on every single image. From these 135 keypoints, 25 refer to body and foot keypoints, which could be used to detect the human position. The others are located on the hands and on the face. More information can be found in the official GitHub website [19].

AlphaPose is an opensource project that has been trained on COCO and MPII dataset and has obtained really good results on the evaluation process. It achieves a result of 75 mAP on COCO dataset and 82.1 mAP on MPII dataset. More information about this project can be found in the GitHub main page [20]

2.4.2. Open-source Python libraries

There are several Python libraries that have been created to work on neural networks fields. These libraries implement a lot of methods that help the user to set a NN architecture, train it and test it. The most famous ones are OpenCV, Keras, Caffe, PyTorch, TensorFlow and Mxnet. Some of them are published online and can be modified by the community. E.g., we could create a new function for one of these libraries and commit it. If it was accepted by the persons in charge, it would be added to the library and everyone would be able to use it.

Some of these previously mentioned libraries include pre-trained models. PyTorch, from Facebook, and TensorFlow, from Google, contain a couple of models that could be useful for our project. These are, keypoint_rcnn50_fpn model from PyTorch and PoseNet from Torchvision.

PyTorch model can be obtained in a Python environment with a couple of coding lines and it's ready to use. It is very accessible since there are several tutorials that guide the user on how to use it. We can store the model in a variable and when we input an image to that variable, we already obtain a dictionary with information about the human boxes and body keypoints. The input image does not even have to be resized and normalized because this function is already performed within the downloaded model.

PyTorch is trained on COCO dataset and, therefore, it returns a total of 17 keypoints. Its performance is 54.6 AP for box detection and 65.0 AP for keypoint detection which are not bad, but neither really good. As it is built on a ResNet50 backbone, which is a large sized backbone, its speed is not good enough for real-time inferences.

Tensorflow is the direct competence of PyTorch. It also provides a model named PoseNet that is particularly thought for edge devices like browsers or mobile phone apps. TensorFlow provides an example App that is capable to load a PoseNet model and take and process images in real-time with the camera. More information about the App is described in [21] and an image of its output is depicted in Fig. 2.7.

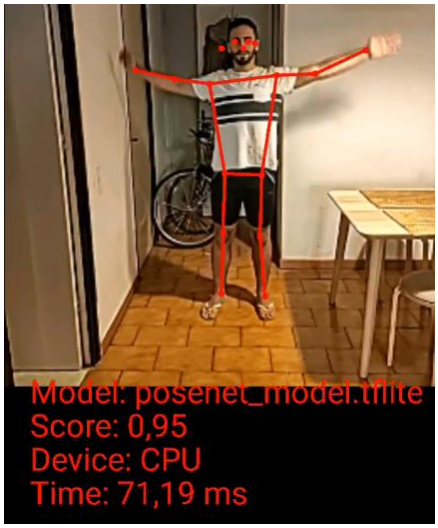


Fig. 2.7. Example of PoseNet App output.

There are several PoseNet models that can be found in [22]. We can choose among these different models but we have to have in mind that there's an accuracy-speed trade-off depending on some parameters that can be set. These parameters are the output stride, input resolution, quantization bytes and multiplier and are represented in the following Fig. 2.8.

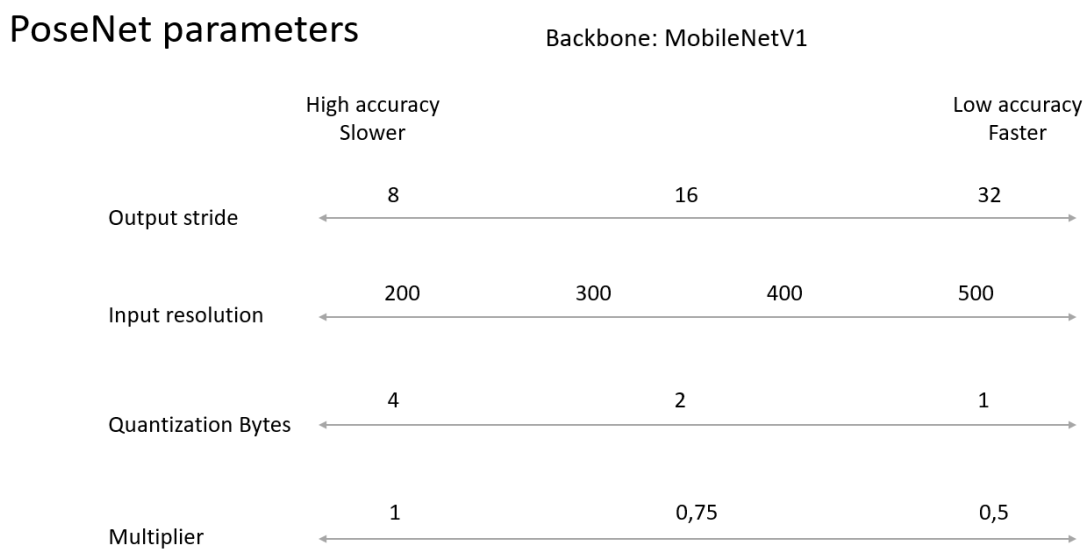


Fig. 2.8. PoseNet parameters affect speed-accuracy trade-off.

2.4.3. Open projects

There are some open-source projects that state that they can be used even for commercial activities. However, either they do not include a lot of information or their performance is not good enough for our purposes.

Tf-pose-estimation is a project that includes some models that have been developed with the intention to be run on edge devices. Its models come from OpenPose, described in 0. Its license says that these models are open and usable even for commercial purposes. However, we doubt about its legacy since OpenPose license stays that commercial usage is forbidden. More information can be found in the official GitHub [23].

PoseEstimationForMobile is an open-source project that implements a Convolutional Pose Machine (CPM) model and Hourglass model using TensorFlow, i.e., they use a bottom-up architecture to locate the keypoints. It has been trained using AI_challenger dataset. It also provides an Android App that can run the model in a mobile phone device. The speed seems to be prominent but, however, the accuracy is not really good. The project is available in GitHub [24].

2.4.4. Private companies

Wrnch is an example of private company that has developed its own computer visions products. As we can see in their official website, one of their technologies is capable to detect human pose detection on edge devices. Naturally, models from private companies are not publically available and their usage would require a payment.

2.5. Evaluation Functions/Metrics

Evaluation Metrics are used to quantify model performance. I.e., according to its result we can guess if a model has been properly trained or not. Its procedure is the following, evaluation images are used as input to the model and the result is compared to the ground truth values. E.g., we input an image to the trained model, we obtain the predicted keypoints and then, we compare them with the original keypoints.

There is not a single way to compare the estimated keypoints with the ground truth ones. In this section we will describe OKS, used in COCO, PDJ, PCP and PCK. We will also talk about the Precision-Recall curve.

2.5.1. Precision-Recall curve

In some metrics the Precision-Recall curve, or PR curve, is used. In order to understand it, we describe the terms Precision and Recall by separate. Precision

is the percentage of correct predictions. E.g., if there are 5 people labelled on an image and we find 10, 5 correct and 5 incorrect, the precision value would be 0.5. Recall value tells us if our model is capable to find all the positives. E.g., if there is an image with 5 people and our model finds 4, the recall value would be 0.8. The following equations 2.1 and 2.2 describe Precision and Recall, respectively.

$$Precision = \frac{TP}{TP+FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.2}$$

Where TP stands for True Positive, FP stands for False Positive and FN stands for False Negative.

When the performance of a model is evaluated, in the first evaluated scenarios the recall value is close to 0 because the number of FN (not found instances) is high and the value of Precision is close to 1 because there are not a lot of FP (incorrectly found instances). The more we progress on the model evaluation, the more incorrect instances are found, decreasing precision, and the more missing instances are found, increasing recall. If we represent the curve of the paired values precision and recall, we obtain the Precision-Recall curve. Some metrics use the area under curve to compute the Average Precision value, known as AP. An illustrative example is represented in Fig. 2.9.

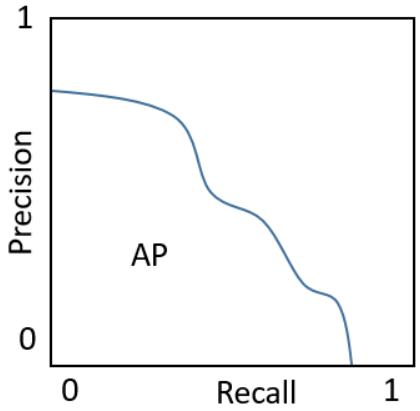


Fig. 2.9. Example of PR curve.

2.5.2. OKS

OKS stands for Object Keypoint Similarity and it's the metric used by COCO dataset. Nowadays, this metric has become the most popular one in keypoint estimation studies. Papers use this metric to compare each other in order to see which performs the best. The explanation of its usage and computation can be found in COCO official website [12] and in the keypoints Challenge presentation.

For each object in each evaluation image, the OKS value is computed according to the following equation 2.3.

$$OKS = \frac{\sum_i \left[\exp\left(-\frac{d_i^2}{2s^2 k_i^2} \right) \delta(v_i < 0) \right]}{\sum_i \delta(v_i > 0)} \quad (2.3)$$

The d_i value is the Euclidean distance between the ground truth keypoint and the estimated keypoint. Value v_i is the ground truth keypoint visibility tag which is 0 if it has not been labelled, 1 if it's labelled and not visible and 2 if it's labelled and visible. Hence, not labelled keypoints do not affect the OKS value. As we can see, d_i is included in an unnormalized Gaussian with standard deviation sk_i. The value of s comes from the total image area divided by the subject bounding box area. The value of k_i is a per-keypoint constant value that controls the Gaussian fall off, its values are seen in the following Table 2.2.

Table 2.2. Keypoint constants for OKS calculations.

Keypoint	k_i
Hips	0.107
Ankles	0.089
Knees	0.087
Shoulders	0.079
Elbows	0.072
Wrists	0.062
Ears	0.035
Nose	0.026
Eyes	0.025

These values have been calculated by the COCO team and they come from the labelling error. They have noticed, that when a person's keypoints are labelled manually, even humans do not always tag the same ground truth location at the same point. That's why there is more tolerance, as we can see in Table 2.2, of hips than of the eyes.

Once the OKS value has been computed for an object, we have to decide if it's good enough to be considered a True Positive or a False Positive. COCO sets several thresholds from 0.5 (loose) to 0.95 (very strict) with a step size of 0.05. If the value is over the threshold, it is considered as a TP, if it's not, it is considered as a FP. If a person within an image has not been found, it's considered as a FN. The Precision-Recall curve can be computed from TP, FP and FN values. We can see a couple of examples of PR curves in the following Fig. 2.10 considering two different thresholds.

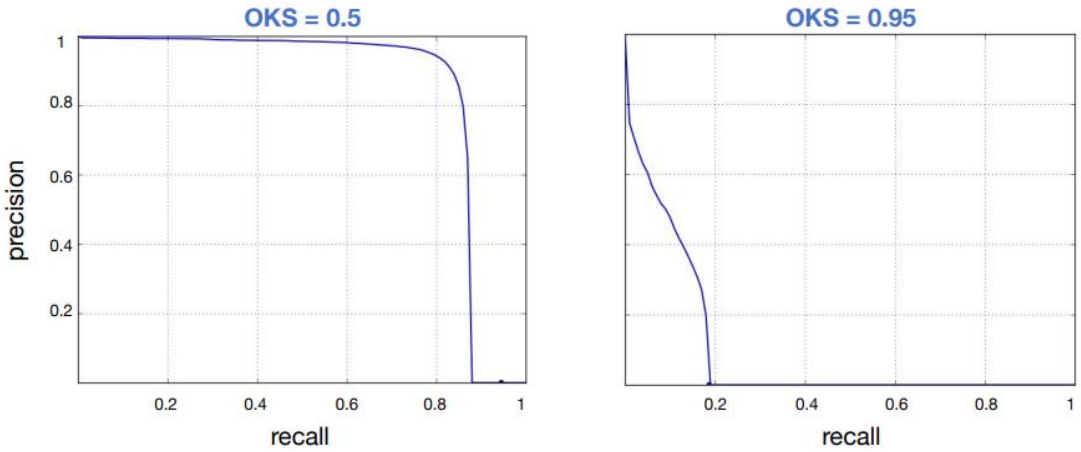


Fig. 2.10. PR curves when using different threshold.

As we can see, the lower the threshold is, the more tolerant it becomes and therefore, the more TP are found. If we compute the area under the curve, we obtain the AP for a specific threshold. If we average all the AP values from 0.5 to 0.95, we finally obtain the mAP metric value. This metric value is used to define the winner of the COCO Keypoints Challenge.

Sometimes mAP and AP terms generate confusion. COCO uses these two expressions with the same meaning. I.e., if we see the word AP with no reference to any threshold, it's because it's referring to the mAP.

COCO also provides a function "analyse()" that returns 180 plots and a report of the model performance. The code is available in GitHub and we can call it from our own computer machine.

2.5.3. PDJ

PDJ stands for Percentage of Detected Joints and is used as an evaluation metric. It checks how many keypoints are estimated within a specific area. This area comes from the ground truth keypoint and the diagonal of the subject bounding box. An example can be seen in Fig. 2.11. If the left keypoint is detected within the orange circle, it will be perceived as a correctly detected keypoint.

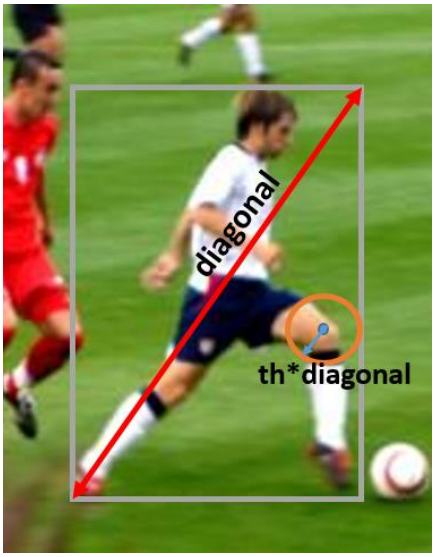


Fig. 2.11. PDJ metric example.

Therefore, PDJ equation is defined as follows in equation 2.4. The closer it gets to $PDJ = 1$, the better estimation the model has performed.

$$PDJ\ @0.05 = \frac{\sum_{i=1}^{n} bool(d_i < 0.05 * diagonal)}{n} \tag{2.4}$$

Where $diagonal$ stands for bounding box diagonal, i.e., the distance from two opposite extremes of the subject bounding box. d_i stands for distance between the ground truth keypoint and the estimated keypoint. n stands for number of keypoints. The value 0.05 can be modified as it acts like a tolerance threshold. The higher it is, the more tolerance is accepted.

In order to compute the mean Average Precision from the PDJ, we compute the PDJ with different threshold tolerances and then average the results. I.e., we can use the following equation 2.5 in order to obtain a mAP for each image.

$$mAP = \frac{\sum_{i=1}^{n} PDJ@th_i}{n} \tag{2.5}$$

Where:

$$th_i = [0.05, 0.10, 0.15, 0.20, 0.25]$$

We have studied the resulting mAP from PDJ for some of our trained models and we have reached the conclusion that a proper PDJ that we should accept as a good result should be a value very close to 1. As we can see in Fig. 2.12, even if the mAP seems to be high, like in the bottom-left image with a value of 0.92, the result is not good enough. In this particular case, the head is estimated to be close to the right knee.

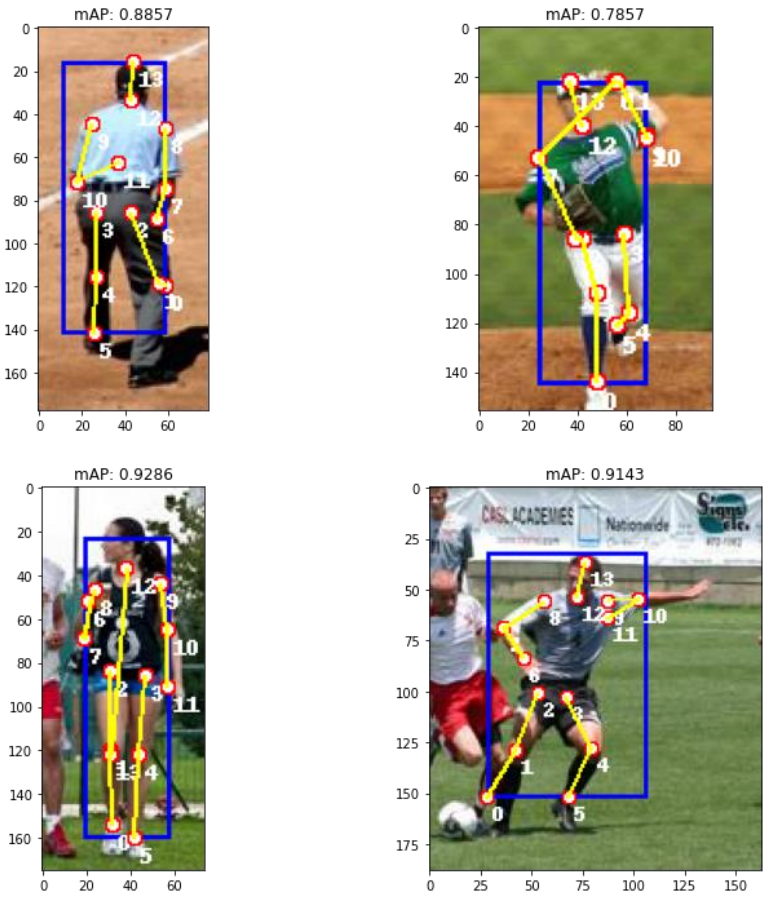


Fig. 2.12. mAP computed from PDJ in some LSP evaluation images.

2.5.4. PCP and PCPm

PCP stands for Percentage of Correct Parts and it analyses how well limbs are estimated. Limbs are considered to be the torso, upper-leg, lower-leg, upper-arm, lower-arm, head, upper-body and lower-body. Each one of these limbs is built from two keypoints located in its corners. In order to compute PCP, the distance from the two ground truth keypoints is computed. If both estimated keypoints are within the 50% distance between the segment endpoints, the limb is considered to be correctly estimated. A clear explanation can be found in Fig. 2.13.

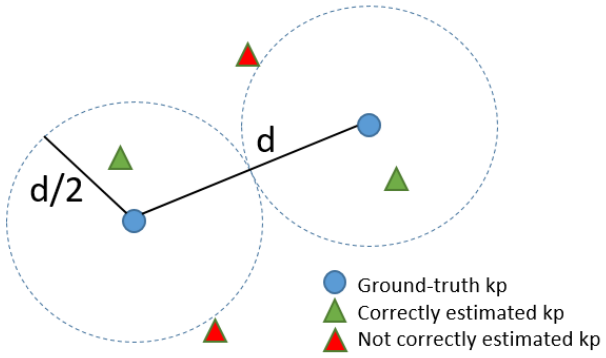


Fig. 2.13. PCP correct and incorrect keypoints.

As explained in [25], PCP is not fair enough since shorter limbs require better precision. That's why, [25] introduces PCPm that instead of considering each individual image limb distance, it computes the mean limb distance over all the images in the test dataset. I.e., there's a different permissive distance for each limb computed from the average distance of that limb in all images.

2.5.5. PCK and PCKh

PCK stands for Percentage of Correct Keypoints and analyses how close keypoints are estimated compared to the ground truth keypoints. It is calculated in a similar way that PDJ does since it uses the person bounding box. However, instead of creating a circle around the ground truth keypoint, it creates a squared box. Therefore, distances are not computed. The usage of this metric is explained in [26], which is the first study to use such measure. Example can be seen in the following Fig. 2.14.

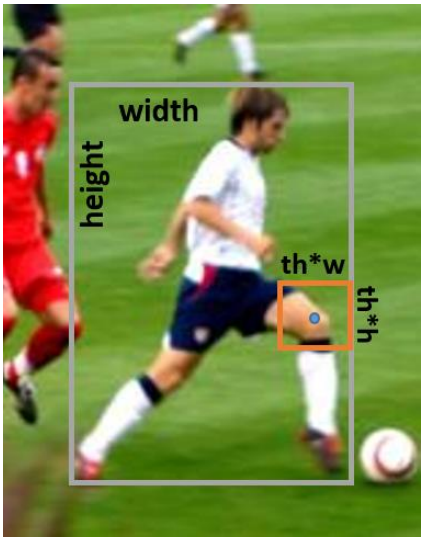


Fig. 2.14. PCK metric example.

The study [25] has created the metric PCKh. It is slightly different from the PCK metric. It uses the 50% of the head limb length, distance between the top of the head and the neck, as the matching threshold. I.e., the threshold represented in Fig. 2.14 is equal to the half of the head limb length.

CHAPTER 3. METHODOLOGY

In this chapter we will explain what we have developed. We will first see what variables and hyperparameters we have had to cope with. We will also present some characteristics of the chosen model, LSP dataset. Then, we will talk about the fine-tuning process that includes information about the chosen model, the chosen evaluation function, freezing configurations and image augmentation.

3.1. Variables and hyperparameters

There is not a single way to compute a fine-tuning on a pre-trained model with a certain dataset. There are multiple variables and hyperparameters to choose from and to be set before training the model. Depending on these parameters, we can end up with a useful model that predicts all our keypoints or with one that cannot.

There are two terms we must get confused with: model hyperparameters and model parameters. A model hyperparameter is set manually by the developer before the model starts training, it is not learnt from the training. E.g., the initial learning rate value is an hyperparameter that we can set. On the other hand, model parameters are variables inside a network that can be trained along the training execution. E.g., weights on a Neural Network are model parameters. In this section, we are referring to model hyperparameters since they are the ones we can tune. When we talk about variables we are referring features that are not related to the training process, like the dataset and the pre-trained model.

Some variables and hyperparameters can be chosen from data analysis, others by studies developed by other research teams and others are chosen by trial-error. For example, it is convenient to choose image augmentation probabilities after performing an analysis of the dataset.

In the current project, we have had to cope with the list of variables and hyperparameters presented in Fig. 3.1 with some examples. Each one of them will be explained in detail in the following sections.

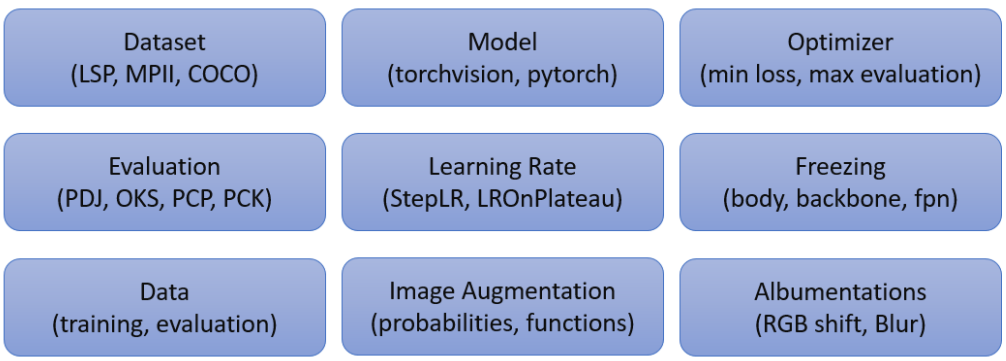


Fig. 3.1. Variables and hyperparameters that must be chosen before starting a fine-tuning process.

3.2. Dataset study

In this project we have chosen LSP dataset as our fine-tuning training dataset because of the following reasons:

- Costless for commercial-usage
- People practicing sports
- Just one person per image.

The LSP dataset provides a file that explains the meaning of each one of the tagged keypoints. Details about LSP keypoints topology can be found in section 2.1.2. Persons' keypoints are registered with coordinates x and y where (0,0) is the top-left corner, (x_{max}, 0) is the top-right corner and (0, y_{max}) is the bottom-left corner.

3.2.1. Visibility tag

Persons' keypoints also include information about their visibility. This field is not described in the information file but we have concluded that visibility tags mean:

- 0: labelled and visible
- 1: labelled and not visible

As we can see in the following Fig. 3.2, not visible keypoints are marked with blue dots and visible keypoints are marked in red.

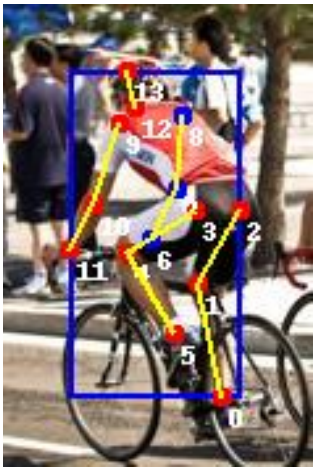


Fig. 3.2. Visible (red) and not visible (blue) keypoints.

The pre-trained model that we have chosen in this project, imported from torchvision, was trained following COCO instructions. Therefore, visibility tags were defined in the COCO way, that is:

- 0: not labelled (kp outside of the image)
- 1: labelled but not visible
- 2: labelled and visible

As we can see, visibility tags are differently defined for COCO dataset and LSP dataset. We have decided to modify our kp visibility according to COCO rules. Then, we have changed all visibility tags equal to 0 into a value of 2. All those keypoints that had a value of 1 have kept such value because the meaning in COCO and LSP remains the same.

3.2.2. Person's box

In order to fine-tune the torchvision model, we are required to provide a bounding box that marks out the area of the person. In COCO dataset this information is provided within the labelled images. However, in LSP dataset no information about the bounding box is provided. Therefore, we have decided to define it as follows:

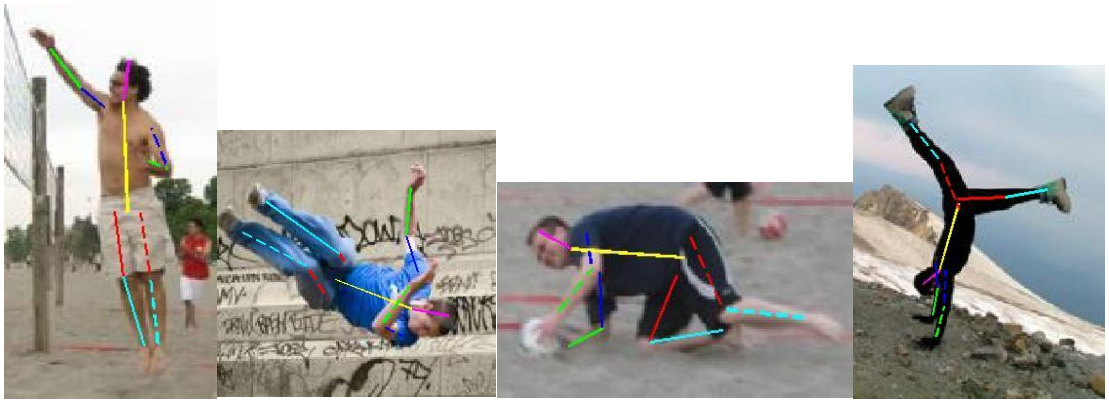
- Top-left coordinate = $[x_{min}, y_{min}]$
- Bottom-right coordinate = $[x_{max}, y_{max}]$

However, after performing some fine-tuning trainings we have realised that the boxes are too narrow to the body. Therefore, we have added 10px extra per size, ending up with the following schema:

- Top-left coordinate = $[x_{min} - 10, y_{min} - 10]$
- Bottom-right coordinate = $[x_{max} + 10, y_{max} + 10]$

3.2.3. Body orientation

As we want to detect keypoints on people's bodies that are practicing sports, we have studied people's orientation in LSP dataset. We want to detect how many people are standing, rotated to the right or to the left or looking downwards. An example of each body rotation can be seen in Fig. 3.3.



(a) (b) (c) (d)

Fig. 3.3 (a) Standing pose (b) right-pose (c) left-pose (d) looking downwards

In order to study the orientation of the bodies we have created a new keypoint, keypoint 14. This one is located in the pelvis, i.e., between keypoint 3 (left hip) and keypoint 4 (right hip). For this purpose, equation 3.1 has been used.

$$kp_{14} = \frac{kp_3 + kp_2}{2} = \left(\frac{kp_{3_x} + kp_{2_x}}{2}, \frac{kp_{3_y} + kp_{2_y}}{2}\right) \quad (3.1)$$

In order to classify each body position into standing, looking left, looking right or looking downwards positions, we have computed the body rotation angle, or torso angle, as shown in Fig. 3.4. Alpha angle has been computed according to equations 3.2 and 3.3. Its value has a value in between -180º to +180º. When "standing", this angle will be close to 0º, when "looking left" it will be negative.

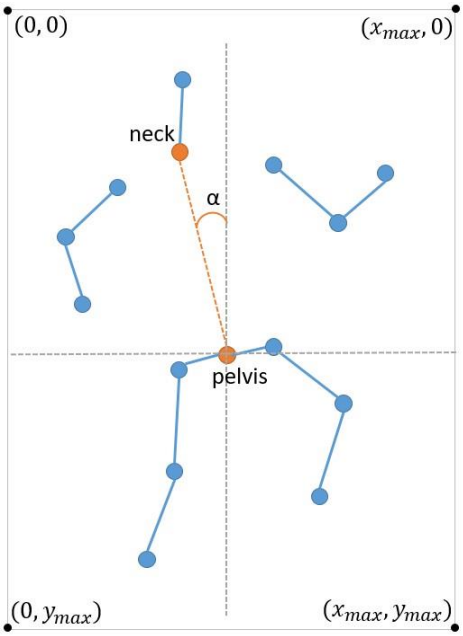


Fig. 3.4 Torso angle is computed from neck and pelvis keypoints.

$$\propto \left(kp_{neck}, kp_{pelvis}\right) = \begin{cases} -90 - \arcsin\left(\frac{kp_{neck_y} - kp_{pelvis_y}}{d}\right), & kp_{pelvis_x} \geq kp_{neck_x} \\ 90 + \arcsin\left(\frac{kp_{neck_y} - kp_{pelvis_y}}{d}\right), & kp_{pelvis_x} < kp_{neck_x} \end{cases} \quad (3.2)$$

Where:

$$d = \sqrt{\left(kp_{neck_x} - kp_{pelvis_x}\right)^2 + \left(kp_{neck_y} - kp_{pelvis_y}\right)^2} \quad (3.3)$$

The following angle values, shown in Table 3.1, have been set to define in which body position a person is located.

Table 3.1. Classification of body positions according to alpha angle.

Position	Degrees
Standing	$-30º \leq \propto \leq 30º$
Looking left	$-150º < \propto < -30º$
Looking right	$150º > \propto > 30º$
Looking downwards	$180º > \propto > 150º$ $-180º < \propto < -150º$

We have applied the equations 3.2 and 3.3 to all the images contained in LSP dataset and we have obtained the following histogram shown in Fig. 3.5.

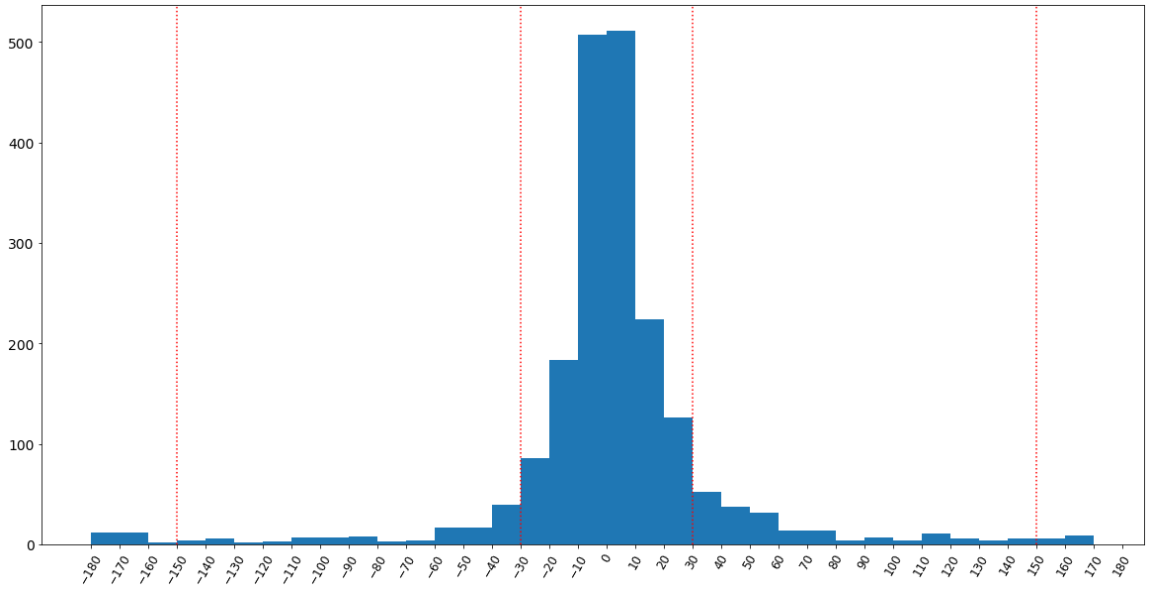


Fig. 3.5. Body orientation histogram.

We have grouped all the image degrees according to table X(before) and he have computed the probabilities of obtaining each one of the different body positions. We can see the results in the following Table 3.2.

Table 3.2. Probabilities of body position in LSP dataset.

Position	Probabilities
Standing	81.85 %
Looking left	5.85 %
Looking right	9.5 %
Looking downwards	2.8 %

As we can see in the previous figures and tables, almost all body positions are standing. This may cause us problems when trying to estimate body positions that are not in a standing position since the model will learn mostly from this kind of data. We will try to correct this result with image augmentation explained in chapter 3.3.4.

3.3. Fine-tuning

3.3.1. Chosen Model

We have decided that we will perform a fine-tuning on the PyTorch model named keypointRcnn_ResNet50_fpn described in section 0. This model uses a top-down architecture, i.e., it first estimates the human body box and then detects the keypoints within the proposed box. We will now present what mistakes this model makes and we will explain “evaluation” and “training” modes. Depending on the mode, the model will expect different inputs and will return different outputs.
Previous to the training and evaluation, we have substituted the last layer of the original architecture. Now, instead of having an output of 17 keypoints, the model will return just 14 because that’s what LSP dataset requires. The new layer is created with random values, i.e., the whole network will keep the pre-trained values but this last layer will not.

3.3.1.1. Main issues

A priori, we have analysed this model and we have realised that its outputs are not outstanding. When the human pose is standing, keypoints are in general well located but, sometimes, we can see some swapping and jittering defects. Moreover, when the human body orientation is not standing and there are body-occlusions, keypoints detection becomes pretty bad. We can see a couple of examples of these two problems in the following Fig. 3.6.

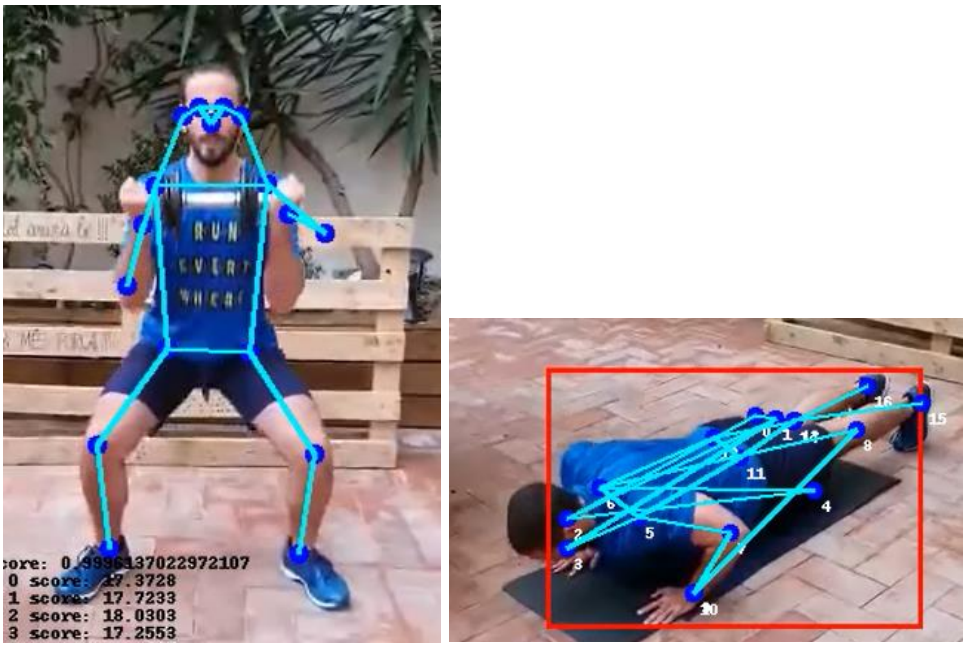


Fig. 3.6. (Left) Jittering error on left elbow. (Right) Keypoints accuracy is lower when the human body is not in a standing position.

3.3.1.2. Training mode

When the training mode is set, we have to provide two inputs to the model. The first one is a list of tensors, each one containing a single image. Images can have different sizes. The second one is a second list of dictionaries that contain the ground truth information about the images. These dictionaries must contain information about the bounding box location, a list that contains the keypoints locations and labels, which in this case are just humans or background. The output is a dictionary that contains five tensors with classification and regression loss values. Next, we can see an example of an output:

{
'**loss_classifier**': tensor(0.0166, device='cuda:0', grad_fn=<NllLossBackward>),
'**loss_box_reg**': tensor(0.0525, device='cuda:0', grad_fn=<DivBackward0>),
'**loss_keypoint**': tensor(2.5782, device='cuda:0', grad_fn=<NllLossBackward>),
'**loss_objectness**': tensor(0.0013, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>),
'**loss_rpn_box_reg**': tensor(0.0022, device='cuda:0', grad_fn=<DivBackward0>)
}

Depending on the specific training purpose, we have used different cost functions. I.e., we have used different combinations of losses that we have tried to minimize. We have set three different training functions. The first one sums up all the losses that appear above and it is used when the whole model is fine-tuned. The second one focuses on keypoints, i.e., it just uses the loss generated by the keypoints, which is normally the highest. We use it when we want to train just the final layers. The third training function that we have used takes only into account the loss generated by the RPN predictor which is loss_rpn_box_reg, as, in some occasions, we have tried to train just the box predictions.

3.3.1.3. Evaluation mode

When the evaluation model is set, we just have to give the model a list of tensors that contain the images. In this mode, the model will return a list that contains a dictionary with the predicted results for each one of the images. The dictionary contains information about:

- Boxes: a list with the predicted boxes with a format of [x1,y1,x2,y2] that points-out the top-left and bottom-right corner.
- Keypoints: a list in which each element is a 3D list containing the keypoints location in a [x,y,v] format where v stands for visibility.
- Labels: predicted labels, that will be 0 or 1. 0 for background and 1 for people.
- Scores: a value that denotes how sure the model is about a certain prediction. We receive the scores of every detected keypoint and also a general score.

We have to note that if the output dictionary returns a list is because it takes into account that there might be more than one person in the image.

3.3.2. Chosen evaluation function

We have seen studies that use LSP dataset with PDJ evaluation function like [27] and others that use LSP together with PCP and PCK evaluation functions like [25]. The evaluation metric that is used in COCO dataset, which is OKS, can't be used in LSP dataset since the keypoints do not match, COCO has 17 and LSP 14. We have ended up choosing PDJ evaluation metric for our study.

3.3.3. Freezing configurations

When we are performing a fine-tuning we don't have to retrain all the layers. E.g., if we use a pre-trained model that classifies animals but we are just interested on classifying cats and dogs, we could freeze all the layers in the network but the last layer. Freezing some layers prevents from destroying information that has already been learnt from the original model. Moreover, the more frozen layers, the faster the network trains.

In this project we have used three different configurations. According to the architecture presented in Fig. 1.11, which represents the current network structure, we have decided to freeze layers as shown in Table 3.3.

Table 3.3. Freezing configurations.

	Backbone body	RPN and ROI_heads
Configuration 1	Tune	Tune
Configuration 2	Freeze	Tune

As we can see, in configuration 1 we are performing a larger training since we are training all the layers in the architecture. On the other hand, in configuration 2 we are just training the last layers of the architecture, those that detect the person keypoints from the suggested body. We have also used configuration number 3 in order to see if the network can predict more properly the proposed regions.

3.3.4. Image Augmentation

Data Augmentation is a common technique that modifies the input data before it gets into the model and trains it. This technique is useful when the dataset is not large enough and we want to avoid the model from over-fitting.

In this project we have used a couple of image augmentation techniques, these are Image Flipping and Image Rotation. As we are working with keypoints, we have to modify the image and also its keypoints and bounding boxes. I.e., if we rotate an image ninety degrees to the left, its keypoints and bounding box must also be rotated.

3.3.4.1. Image flipping

Image Flipping is like placing a mirror on the right side of the image. The flipped image is the one represented on the mirror. We can see an example of image flipping in the following Fig. 3.7.

(a) (b)

Fig. 3.7. (b) has been flipped from (a).

In order to rotate the keypoints, we exchange left extremities with right extremities and then, we subtract the x coordinate value from the image width. Head and neck keypoints are not exchanged with any other keypoint. An example of keypoint flipping would be the one represented in equation 3.4.

$$kp_{left\ hip} = \left(width - kp_{right\ hip_x}, kp_{right\ hip_y}\right) \quad (3.4)$$

The visibility tag is also exchanged in the extremities keypoints. E.g., if the left hip was not visible and the right one was, from now on, the left hip will be visible and the right one will not. I.e., the visibility tag is also switched.

3.3.4.2. Image Rotation

In section 3.2.3. of this document we have described body orientation and we have analysed LSP dataset. Our conclusion has been that the majority of the subjects are in a standing position. This may cause problems to our model when trying to estimate keypoints of people that are not in such position. We want to solve this problem by applying an Image Augmentation technique called Image Rotation. Our objective is to flatten the histogram in Fig. 3.5. in order to train a model capable to detect keypoints independently of the body position.

In this project, we rotate images 90 degrees to the left, 90 degrees to the right or 180 degrees with a certain probability that is set before training the model. In the next section we will give more details about the chosen probabilities.

The study [28]suggests that, when we want to obtain the keypoints from a video or image, if we know the body orientation, we can rotate the image before processing it by the model, so the body ends up in a standing position, and then, the model will return a better accuracy. However, we have tried this method and the results have not improved enough. Moreover, we prefer training a model that can directly predict body keypoints without previously interfering on the image.

If we want to rotate images, we also have to rotate keypoints and bounding boxes. In order to do so, we have used equations 3.5, 3.6 and 3.7 to rotate keypoints when rotating 90º to the left, 90º to the right and 180º, respectively. We have also used equations 3.8, 3.9 and 3.10 to rotate bounding boxes when rotating 90º to the left, 90º to the right and 180º, respectively. Width (w) and height (h) stand for the width and height of the image before it is rotated.

$$kp_{new} = (kp_{old_y}, w - kp_{old_x}) \quad (3.5)$$

$$kp_{new} = (h - kp_{old_y}, kp_{old_x}) \quad (3.6)$$

$$kp_{new} = (w - kp_{old_x}, h - kp_{old_y}) \quad (3.7)$$

$$box_{new} = (box_{old_1}, w - box_{old_2}, box_{old_3}, w - box_{old_0}) \quad (3.8)$$

$$box_{new} = (h - box_{old_3}, box_{old_0}, h - box_{old_1}, box_{old_2}) \quad (3.9)$$

$$box_{new} = (w - box_{old_2}, h - box_{old_3}, w - box_{old_0}, h - box_{old_1}) \quad (3.10)$$

Where the old box is defined as follows:

$$box_{old} = (box_{old_0}, box_{old_1}, box_{old_2}, box_{old_3}) \quad (3.11)$$

As we can see, box is defined by four coordinates. The first two ones refer to x and y of the left-top corner of the box and the last two ones refer to x and y of the bottom-right corner of the box.

The result of image rotation can be seen in the following Fig. 3.8. As we can see, the image has been properly rotated within its keypoints and bounding box.

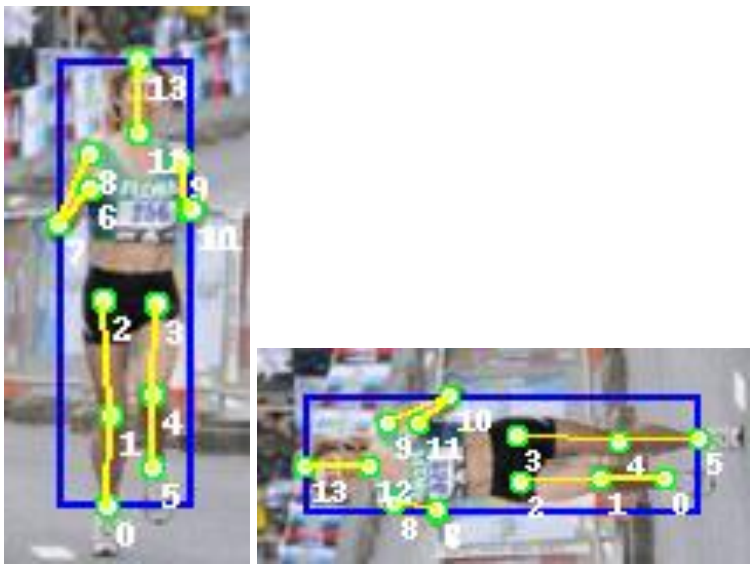


Fig. 3.8. Original (left) and rotated (right) image.

3.3.4.3. Probabilities

In this project we have used different values of rotation probabilities that will be explained in this section. As we will later see in the results section 4, the model learning depends a lot on these values. We have created four different configurations shown in Table 3.4.

Table 3.4. Sets of Image Augmentation Probabilities.

	Rotate left (%)	Rotate right (%)	Rotate 180º (%)	Flipping (%)
Null	0	0	0	50
Low	5	5	5	50
High	20	20	10	50
Uniform	21	22	28	50

According to the selected set of probabilities, the chances of finding a human pose that has a torso looking upwards, looking downwards, right or left will change. We can see the distribution of the torsos orientations depending on the assigned probabilities in Fig. 3.9 and Fig. 3.10. We can notice that the higher the probabilities, the more flattened are the chances.

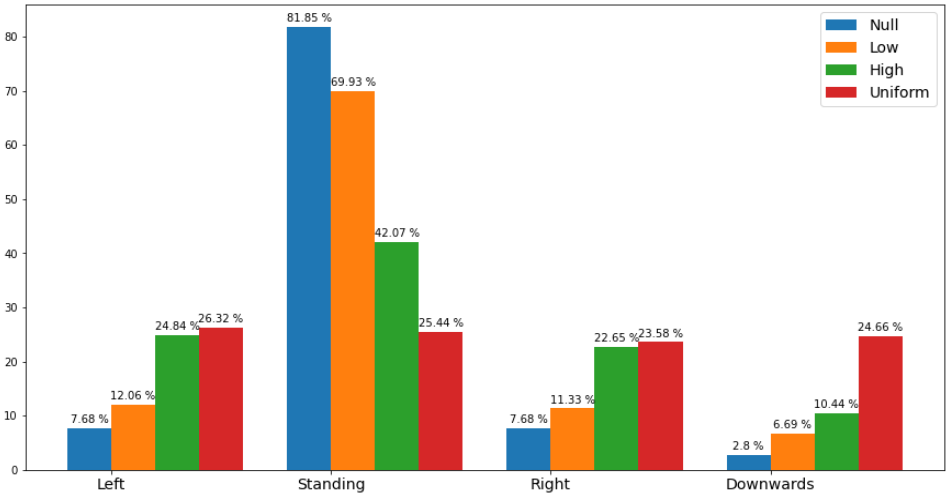


Fig. 3.9. Distribution after image augmentation.

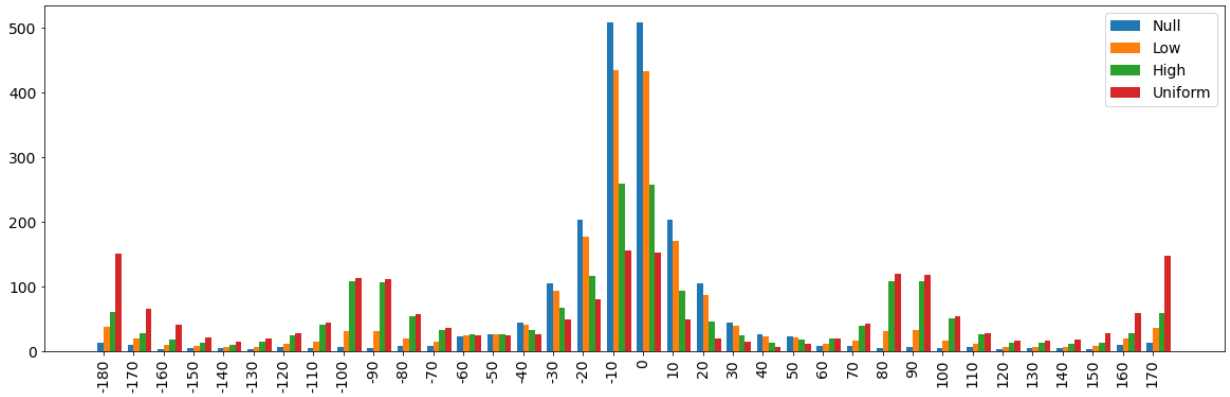


Fig. 3.10. Distribution after image augmentation classified by degrees.

As we can see, uniform distribution (red) has been adjusted in order to have similar probabilities of obtaining a person in a right, left, standing and looking downwards orientation.

3.3.5. Process

The fine-tuning process that we have developed follows the structure depicted in Fig. 3.11. The tutorial presented in [29] has been followed for the realisation of this part of the project. As we can see on the top of the image, we start by downloading the resources that are needed like the LSP dataset, the PyTorch pretrained model and the evaluation and training functions.

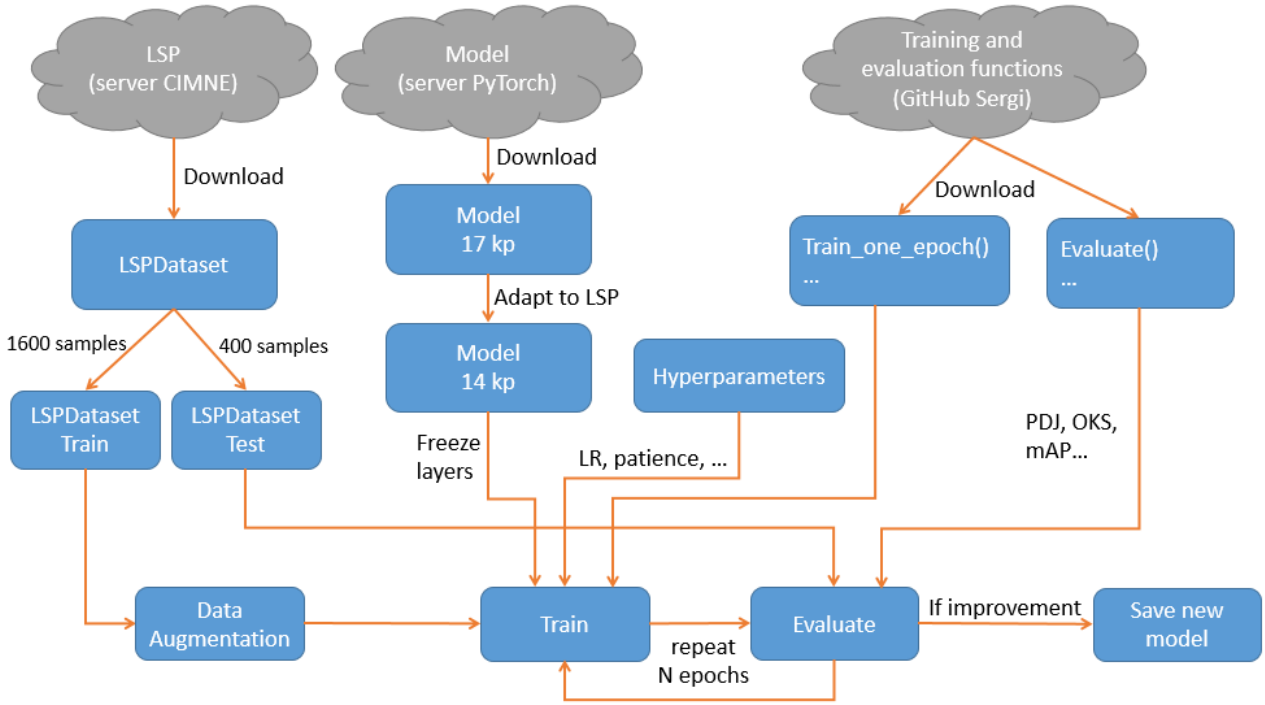


Fig. 3.11. Fine-tuning process scheme.

Once the dataset has been downloaded, we have to create a dataset class that inherits from the standard “torch.utils.data.Daset”. This class must contain, at least, a couple of functions “__len__” and “__getitem__”. This second function must return the image in a PIL format and a dictionary that contains information about such image, for example, the keypoint and the box location. These values will be used as ground truth during the training process.

Then, we have divided the dataset in two parts; training dataset and evaluation dataset. We have decided to use 1600 randomly selected images in the training dataset and 400 in the evaluation dataset. The images that are contained in the training dataset will have a probability to be transformed by the Image Augmentation functions. Images in the evaluation dataset will not be altered by

the IA techniques. We have also set a batch size of 2, i.e., model weights will be updated after 2 images have been processed.

The model that we download is ready to detect 17 keypoints on the human body. However, LSP dataset contains just 14 keypoints. Hence, we have had to modify the last layer of the model in order to return a total of 14 keypoints instead of 17. Before the training starts, we also have to decide which layers of the network will be frozen, i.e., which layers will keep their original values and won't be trained.

Then, we have to take some decisions about the used learning rate. We must choose in between one of the functions named in section 1.4.2 and, according to the chosen algorithm, we will also have to choose some hyperparameters. E.g., if LR on Plateau is used, we must specify the patience value described in section 1.4.2.2. Moreover, we have to execute the optimal LR finder before the training starts in order to find the initial LR value.

As explained in 3.3.1, we can focus on the keypoint loss function, the general loss function or even the box loss function. We select what function we want to minimize in the "train_one_epoch" function.

When all these processes have been loaded in memory, we can start training the model. It will then perform a process like the one explained in section 1.4. The loss returned by the model after every iteration is stored in a list that will be later averaged, obtaining a loss value at the end of each epoch.

Once the model has gone through all the training images, we change its configuration into evaluation mode. Now, the model does not expect a dictionary containing the ground truth information of the images, it just requires the image as an input since it will just predict its result, i.e., it will not update any weight. For every image prediction, we compute the PDJ function described in section 0 and obtain a mAP value. Then, all the images mAPs are averaged obtaining the epoch mAP value.

We repeat the two previous processes a determined amount of times, called epochs. In this project, we have normally used a value of 30 epochs. We save the weights of the model if we see that at the end of the epoch, the mAP value is larger than previous mAP values.

And, last but not least, when the model has been trained, a second evaluation is performed using home-made videos in which I appear practicing different types of exercises. These exercises include keypoint occlusion and are similar to the exercises that the final user will perform. A code has been developed in order to predict and tag the videos. An example can be seen in Fig. 3.12.

Fig. 3.12. Example of a home-made video tagged with keypoints predictions.

All the keypoints scores returned by the model are printed on the image and the frame rate is computed according to the elapsed time on keypoint prediction. We accumulate all the scores and plot them in boxplot diagram like Fig. 3.13.

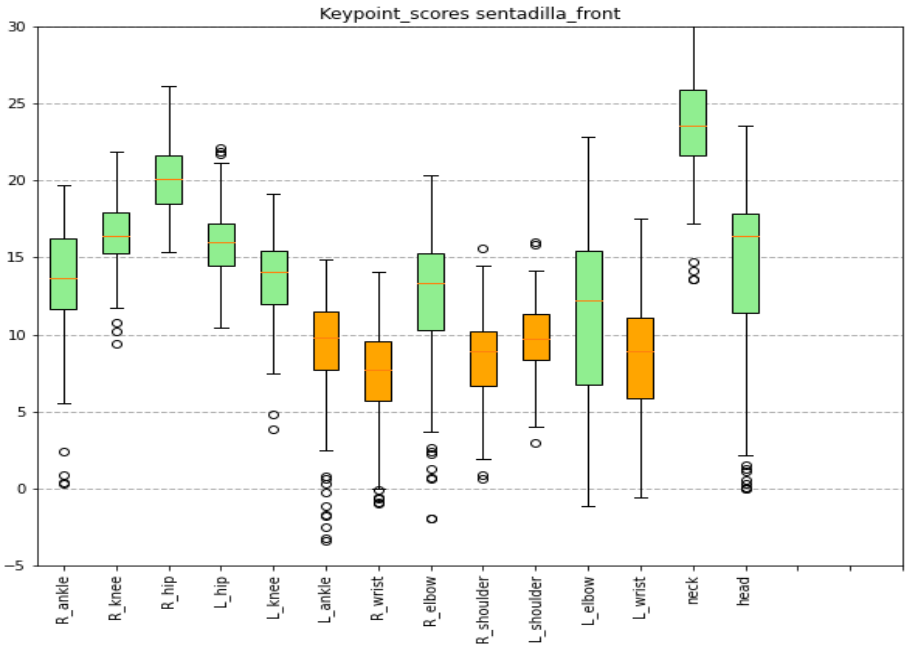


Fig. 3.13. Example of keypoint scores obtained from a video.

CHAPTER 4. RESULTS

As we have seen in chapter 3.1, there are a lot of variables and hyperparameters to choose from. In this section we will present what configurations, combinations of those variables and hyperparameters, we have used in our trainings. When we started this fine-tuning process, we did not know if it was better to fine-tune the whole model or just the last layers. Hence, we decided to evaluate both branches and see what are the differences and what gives us better results.

4.1 Training last layers

We have started with the fast one; training just the last layers in order to see if results are good enough. Therefore, the backbone body and FPN layers will not alter their values, just the ROI_heads and FPN will. The cost function of all these trainings is the one from loss_kp, i.e., we try to reduce the loss of the keypoints. We have then designed the configurations shown in Table 4.1.

Table 4.1. Training configurations that have been used when training last layers.

Train	Optim.	LR	IA Prob.	Freezing	Loss	mAP
1	Loss_kp	StepLR	Null	2	3.170	0.843
2	Loss_kp	CLR	Null	2	2.359	0.817
3	Loss_kp	Plateau (min Loss)	Null	2	2.48	0.843
4	Loss_kp	Plateau (max mAP)	Null	2	2.06	0.840
5	Loss_kp	Plateau (min Loss)	High	2	8.113	0.156
6	Loss_kp	Plateau (min Loss)	Low	2	2.59	0.819
7	Loss_kp	Plateau (max mAP)	Null Uniform	2 2	2.06 3.97	0.843 0.843
8	Loss_kp	Plateau (max mAP)	Null Uniform	2 2	2.06 3.88	0.843 0.834

With the first 4 trainings we have tried to analyse which LR performs the best and we have noticed that LROnPlateau is the one that learns the most. We can appreciate that is capable to reduce the loss and increase the mAP more than StepLR or CLR algorithms. Because of that, from now on, we will mostly use this LR algorithm.

Results of configuration 4 are depicted in Fig. 4.1(Left) and Table 4.2. This model has learnt how to locate keypoints on a human body that is in a standing position with a mAP of 0.874. This is because the 81.85% of LSP images are in a standing position. In order to increase mAP value for non-standing orientations, we have tried to incorporate image augmentation techniques, described in chapter 3.3.4. Then, we have trained models 5 to 8 with different sets of probabilities described in Table 3.4.

We have to remember that we have substituted the last layer of the original model with a new layer that instead of generating 17 keypoints, it generates 14. Therefore, this layer will always be random at the beginning of our trainings. In models 5 and 6 we have started the fine-tuning process with this new randomly generated layer and in trainings 7 and 8 we have started from the obtained result in training 4. The difference between 7 and 8 is the starting learning rate. In 8, the learning rate has been set to a larger value.

Model 5 has faced too high probabilities and, because of that, it has not been able to learn. As we can see, it's evaluation metric mAP has got stuck at a very low value, 0.156. Model 6, instead, has faced lower probabilities and has been able to learn. However, mAP results in model 6 are not good enough.

Models 7 and 8 have been able to learn a little bit from the "uniform" rotation probabilities. As it can be seen in Table 4.2 and in Fig. 4.1, "left" and "down" human body rotation have obtained better results. However, mAP values are still very similar to the ones obtained in model 4. We can conclude, then, that by applying image rotation techniques we can increase a little bit the detection in non-standing positions.

Table 4.2. mAP at the end of the model 4 and model 8 training.

Train	Average	Standing	Left	Right	Down
4	**0.840**	**0.874**	0.713	**0.698**	0.438
8	0.834	0.863	**0.723**	0.697	**0.531**

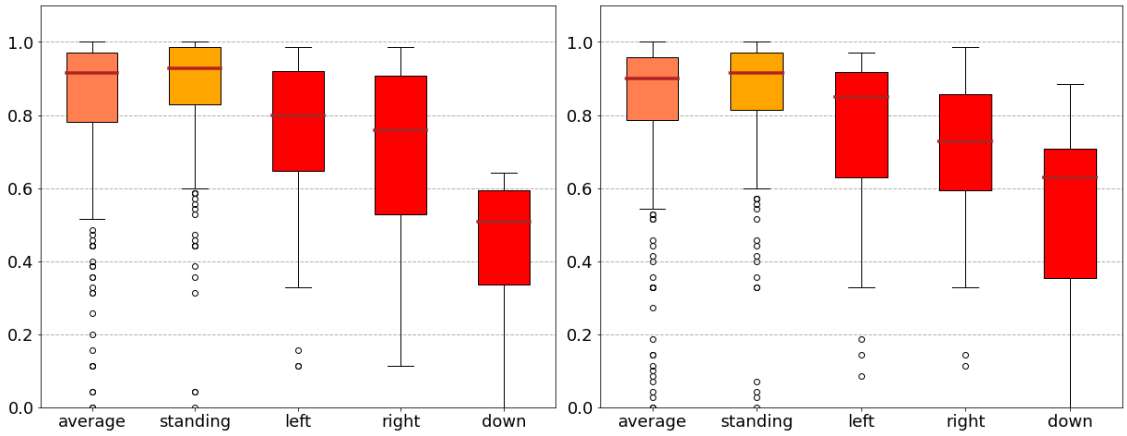


Fig. 4.1. (Left) mAP distribution of model 4 (Right) mAP distribution of model 8.

We have also tested these models with some home-made real case videos and we have concluded that the one that performs the best is model 4. However, results are not much better than the ones from the pre-trained PyTorch model. We can see some keypoint prediction examples of model 4 in Fig. 4.2. If we test model's performance when the human position in not standing, the result is not as good as the one from the pre-trained PyTorch model. We can see a comparison of two outputs in Fig. 4.3.

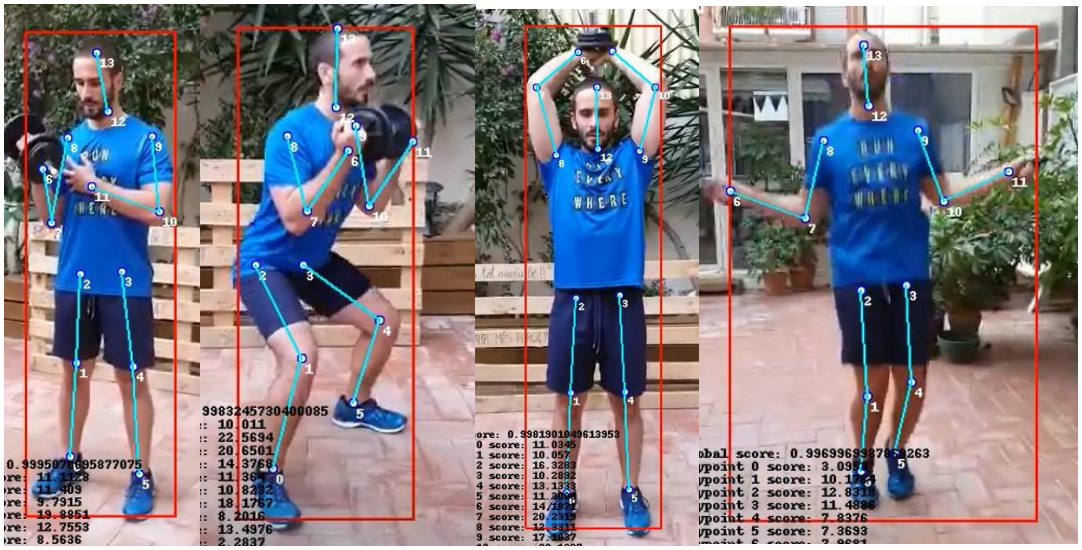


Fig. 4.2. Some examples of model 4 performance when the subject is standing.

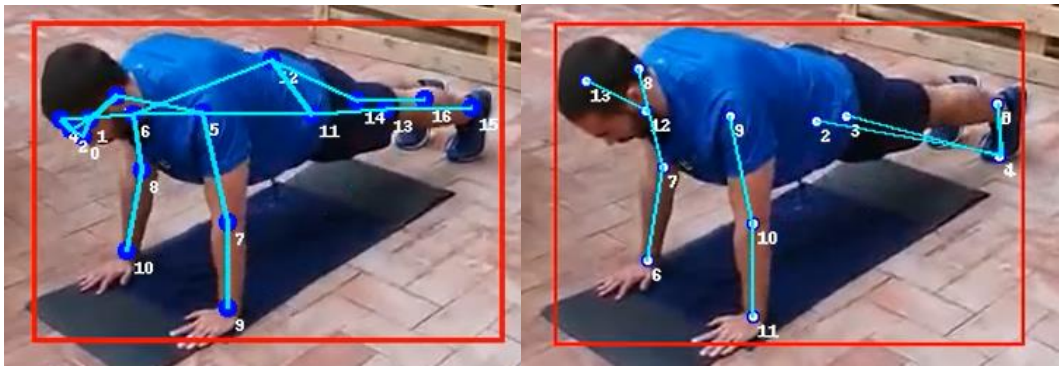


Fig. 4.3. Keypoint prediction on a real case exercise. (Left) PyTorch pre-trained model. (Right) Model 4. PyTorch gets more accurate results.

4.2 Training the whole network

We have also tried to train the whole network in order to see if its performance can get any better. In the following Table 4.3 we can see the sets of configurations that we have used.

Table 4.3. Training configurations that have been used when training all layers.

Train	Optim.	LR Plateau	IA Prob.	Freezing	Loss	mAP
9	Loss_kp	min loss	High	1 2	2.844 1.447	0.818 0.842
10	Loss_kp	mAP	Low	1 2	2.702 2.002	0.883 0.876
11	Total_loss Loss_kp	mAP	High	1 2	3.23 -	0.903 -
12	Total_loss Loss_kp	mAP	Null	1 2	1.963 1.519	0.925 0.918
13	Total_loss Loss_kp	mAP mAP	Uniform	1 2	2.972 2.803	0.930 0.932

As we can see in Table 4.3, in all these scenarios we have performed a couple of trainings for every configuration set. That's why, every row of Table 4.3 contains two rows inside. We have first trained the whole network with a freezing configuration 1 (nothing is frozen) and then, we have trained the obtained model with a freezing configuration of 2 (backbone is frozen).

Training 9 and training 10 have been executed first. They have tried to minimize the keypoint loss; they did not consider the other losses. Results are not good, basically because the person bounding box is not detected properly. As we are adapting the whole network but focusing on the keypoint error, boxes have gotten untrained. If the proposed regions are not accurate and do not include the whole body, keypoints can't be detected. For example, if the box does not include the ankles, the keypoint estimator will never detect them properly since it will not consider that area for the detection. Some examples are shown in Fig. 4.4.

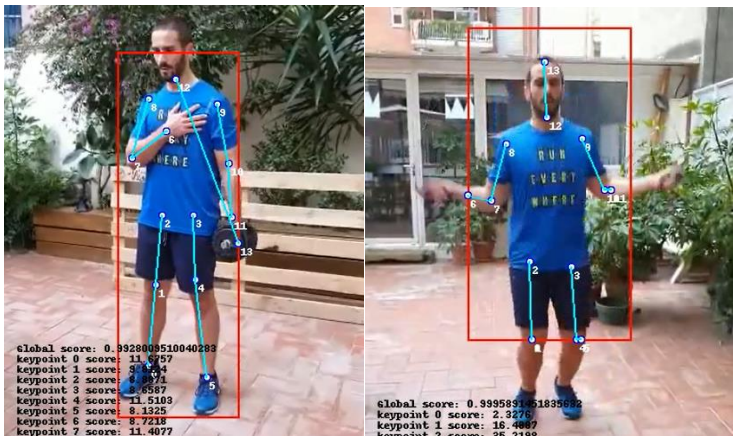


Fig. 4.4. Left example does not include the top of the head in the box. Right example does not include wrists nor ankles in the box.

Then, in train 11, we have decided to consider the whole set of losses. I.e., now the cost function to minimise is the sum of all the losses instead of considering just the keypoint losses. The result, mAP equal to 0.903, is better than anyone previously achieved. When we analyse its results we notice that boxes have been better proposed than in train 9 and 10. Therefore, we conclude that using all the losses improves the output. But, however, we have noticed that bounding boxes are too tight to the human borders and, because of that, some extremities are not included in the proposed box and they can't be properly detected.

In order to solve the problem in train 11, we have decided to increase the size of the bounding boxes 10px in each side. From this correction, we have trained model 12 and 13. Model 12 has tried to learn from a "Null" configuration of probabilities for image rotation; no Image Augmentation apart from image flipping is performed, as described in Table 3.4. On the other hand, model 13 has tried to learn with a "Uniform" configuration. We have obtained results that we never got before reaching a mAP value of 0.932 in the validation dataset, as we can see in Table 4.4 and Fig. 4.5. We can conclude that fine-tuning the whole network is more beneficial than fine-tuning just the last layers. Moreover, we can also conclude that by increasing the IA probabilities, the model learns better how to

detect keypoints in non-standing body orientations. All this, without compromising the detection on standing positions.

Table 4.4. Comparison among the best obtained models. We notice that the one that works better with the validation dataset is model 13.

Train	Training	Average	Standing	Left	Right	Down
12	Whole model	0.918	0.950	0.783	0.763	0.650
13	Whole model	**0.932**	**0.953**	**0.863**	**0.792**	**0.834**
4	Last layers	0.840	0.874	0.713	0.698	0.438
8	Last layers	0.834	0.863	0.723	0.697	0.531

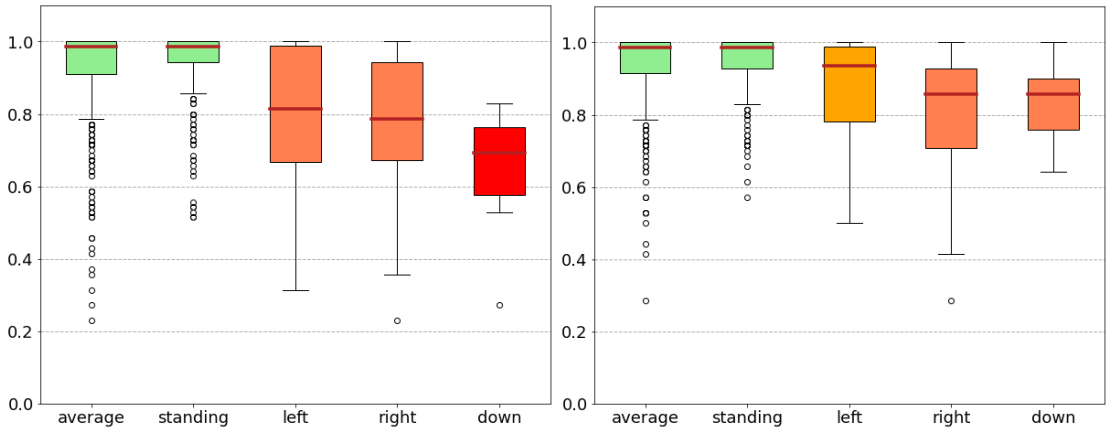


Fig. 4.5. (Left) mAP values of model 12 (Right) mAP values of model 13. By applying IA, the model learns better how to estimate keypoints on “right”, “left” and “down” orientations without compromising “standing” orientations.

However, even if the results on LSP dataset are good, we have noticed that when model 13 is applied to our home-made videos, the inference obtained by the model is not good enough as we can see in the examples depicted in Fig. 4.6. Similar results are obtained when model 12 is used. We can conclude, then, that the better performance of the model with LSP dataset is not strictly correlated with the better performance of the model with our real case examples. Hence, we can state that LSP images do not properly represent our real case images.

Fig. 4.6. Examples of errors produced by model 13 on our real case videos.

CONCLUSIONS

During the development of this project I have learnt a lot of concepts, terminologies and computer vision basis that before were just mere ideas to me. I have the feeling that I have made several steps into a world that I just had a general overview of it.

Keypoint detection is normally used for unconstrained environments, i.e., the training and testing datasets contain images from a lot of kind of scenarios where multiple or single people appear, different sports and activities are performed, people are in different shapes, wear different clothes, etc. We thought that by constraining the environment using a sports dedicated dataset like LSP, in which a single person appears in the middle of the image, we could get a model that properly detected keypoints in sports scenarios.

Performing a fine-tuning technique seemed to be the best option since it's cheap on time; we do not have to spend too much time designing a whole architecture and training it with a huge dataset. Moreover, using PyTorch pre-trained model was a good option since its open-source and well documented.

After performing the trainings described in chapter 4, we have reached several interesting conclusions. First one is that fine-tuning the whole pre-trained model ends up with better results than just training the last layers. Second is that image augmentation techniques are very useful in order to help the model to generalise. And finally, unfortunately, we have concluded that LSP dataset is not correlated enough to our real case scenario. Even if the obtained model can properly predict keypoints on LSP validation dataset, it cannot obtain the same results in our realistic scenario videos.

We have obtained a model, number 4, that can predict keypoints with good accuracy in our real case videos. However, its performance is not better than the one that we can already obtain with PyTorch pre-trained model. Therefore, as we have not obtained the desired goal, we will continue working and doing research in order to obtain a model that suits our requirements.

In terms of data, we could create our own dataset of people practicing similar sports that the one that we will later have to analyse. This option can be very time-expensive and we might not have the tools nor the time to tag hundreds of images, but probably it's worth the time in order to obtain a proper result. LSP images have a really poor resolution that may have affected negatively our final results. Probably if we used images with better resolution, feature extraction layers would perform better.

Inside the model we could change several things like the pre-trained backbone. We could also add some RNN (Recurrent Neural Network) layers since our input images are contiguous one to the other. However, in order to do so, we would need tagged videos as input during the training process instead of images that have no relationship among them. We could probably also change the anchor generator that is related with the RPN. Probably we can redesign it in order to

optimise it to detect a single person in the centre of the image. We could also try to use a different pre-trained model. In such case, we should probably change from top-down model to a bottom-up one, or a model that combined both approaches.

The application that will later run our trained model can also provide some “real-time” corrections, i.e., corrections that are not applied inside the model. There are some situations in which the error is “obvious” and we can easily correct it after the model has given us the wrong result. For example, when right and left ankles are swapped. If we see that one leg is going into one direction, it’s not normal that after the right knee we find the left ankle. We could then correct it in real-time “manually” swapping the results of both ankles. Another possible solution is to treat the image before it gets processed by the model. For example, if the body orientation is looking to the left and the model is better on detecting standing positions, we could rotate the image ninety degrees to the right.

About the general project development, it has been difficult to read, understand, filter and apply the tones of information that there are behind Computer Vision field. What every tutorial teaches as simple, becomes more and more complicated the more you want to work with it. At the beginning of this project we tried to work at a high-level, not getting into neural network theory and trying to avoid difficult programming. We tried to do some study of the art research in order to find a free model ready to be used in an app. However, we realised it was not that easy and the steps we made were always backwards, moving us from the final product to the neural network theory and programming, in order to obtain our own model.

I have realised how difficult it gets when you try to develop a project for a private company. You must have a look at all the licences and cope with the fact that the majority of them state that you cannot use their tools.

I have also faced difficulties with the used programming frameworks. I started using Anaconda and then discovered Google Colab. It is a really powerful tool that can run all your codes on the cloud in really powerful GPUs. However, once my code was all prepared for this programming environment, I realised that it has some computing limitations that I had to cope with. Therefore, sometimes, I have had to wait for a couple of days in order to run my next script.

Our further steps, apart from improving this model as much as we can, are the development of another model that is capable to process images in “real-time” and on-edge devices. We will possibly try to change the current backbone into a MobileNet2 backbone which is much faster but will probably lose accuracy. Then, models can be converted to on-edge device formats in order to be processed in an optimal way on mobile phone or other devices.

About the sustainability considerations, this project will let people to stay and train at home instead of taking a private or private vehicle and transport themselves to a gym or rehabilitation centre. This can reduce the consume of private transportations and, therefore, reduce the emitted greenhouse gases.

In relation to the ethical considerations, this project will make accessible to everyone the possibility of being corrected in real-time when practicing sports or performing rehabilitation exercises. For example, during coronavirus lockdown, people started practicing sports at home. They could use this tool in order to avoid harming themselves when performing exercises in a wrong position. Another example is that this project could help people that have mobility problems. They can now avoid travelling to the rehabilitation centre, with all the implied difficulties, and practice the same exercises at home while being corrected in real-time.

ACRONYMS

AI	Artificial Intelligence
AP	Average Precision
CNN	Convolutional Neural Network
DL	Deep Learning
FCN	Fully Convolutional Network
FC	Fully Connected
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
IA	Image Augmentation
LSP	Leeds Sports Pose
LSPe	Leeds Sports Pose extended
mAP	Mean Average Precision
ML	Machine Learning
NN	Neural Network
OKS	Object Keypoint Similarity
PCK	Percentage of Correct Keypoints
PCKh	Percentage of Correct Keypoints head
PCP	Percentage of Correct Parts
PCPm	Percentage of Correct Parts mean
PDJ	Percentage of Detected Joints
RCNN	Regions with CNN
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
TP	True Positive

REFERENCES

[1] J. Huang *et al.*, ‘Speed/accuracy trade-offs for modern convolutional object detectors’, Apr. 2017.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ‘Rich feature hierarchies for accurate object detection and semantic segmentation’, Oct. 2014.

[3] R. Girshick, ‘Fast R-CNN’, Sep. 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, ‘Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks’, Jan. 2016.

[5] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, ‘Feature Pyramid Networks for Object Detection’, Apr. 2017.

[6] ‘torchvision.models — PyTorch 1.6.0 documentation’. https://pytorch.org/docs/stable/torchvision/models.html#keypoint-r-cnn (accessed Oct. 21, 2020).

[7] ‘torch.optim — PyTorch 1.6.0 documentation’. https://pytorch.org/docs/stable/optim.html#torch.optim.SGD (accessed Oct. 21, 2020).

[8] ‘CS231n Convolutional Neural Networks for Visual Recognition’. https://cs231n.github.io/neural-networks-3/ (accessed Oct. 21, 2020).

[9] L. N. Smith, ‘Cyclical Learning Rates for Training Neural Networks’, Apr. 2017.

[10] T.-Y. Lin *et al.*, ‘Microsoft COCO: Common Objects in Context’, Feb. 2015.

[11] ‘COCO - Common Objects in Context’. https://cocodataset.org/#home (accessed Oct. 21, 2020).

[12] ‘COCO - Common Objects in Context’. https://cocodataset.org/#keypoints-eval (accessed Oct. 21, 2020).

[13] Y. Huang, B. Sun, H. Kan, J. Zhuang, and Z. Qin, ‘FollowMeUp Sports: New Benchmark for 2D Human Keypoint Recognition’, Nov. 2019.

[14] S. Johnson and M. Everingham, ‘Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation’, in *Procedings of the British Machine Vision Conference 2010*, Aberystwyth, 2010, p. 12.1-12.11, [Online]. Available: http://www.bmva.org/bmvc/2010/conference/paper12/index.html.

[15] ‘Leeds Sports Pose Extended Training Dataset’. https://sam.johnson.io/research/lspet.html (accessed Oct. 21, 2020).

[16] ‘MPII Human Pose Database’. http://human-pose.mpi-inf.mpg.de/ (accessed Oct. 21, 2020).

[17] M. Verma, S. Kumawat, Y. Nakashima, and S. Raman, ‘Yoga-82: A New Dataset for Fine-grained Classification of Human Poses’, Apr. 2020.

[18] S. Jin *et al.*, ‘Towards Multi-Person Pose Tracking: Bottom-up and Top-down Methods’, p. 4.

[19] ‘CMU-Perceptual-Computing-Lab/openpose: OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation’. https://github.com/CMU-Perceptual-Computing-Lab/openpose (accessed Oct. 21, 2020).

[20] ‘MVIG-SJTU/AlphaPose: Real-Time and Accurate Full-Body Multi-Person Pose Estimation&Tracking System’. https://github.com/MVIG-SJTU/AlphaPose (accessed Oct. 21, 2020).

[21] TensorFlow, ‘Track human poses in real-time on Android with TensorFlow Lite’, *Medium*, Aug. 12, 2019. https://medium.com/tensorflow/track-human-poses-in-real-time-on-android-with-tensorflow-lite-e66d0f3e6f9e (accessed Oct. 21, 2020).
[22] ‘pose | TensorFlow Hub’. https://tfhub.dev/s?module-type=image-augmentation,image-classification,image-classification-logits,image-classifier,image-feature-vector,image-generator,image-object-detection,image-others,image-pose-detection,image-segmentation,image-style-transfer,image-super-resolution,image-rnn-agent&q=pose (accessed Oct. 21, 2020).
[23] I. Kim, *ildoonet/tf-pose-estimation*. 2020.
[24] edvardHua, *edvardHua/PoseEstimationForMobile*. 2020.
[25] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, ‘2D Human Pose Estimation: New Benchmark and State of the Art Analysis’, in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, Jun. 2014, pp. 3686–3693, [Online]. Available: https://ieeexplore.ieee.org/document/6909866.
[26] Y. Yang and D. Ramanan, ‘Articulated Human Detection with Flexible Mixtures of Parts’, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2878–2890, Dec. 2013.
[27] X. Fan, K. Zheng, Y. Lin, and S. Wang, ‘Combining Local Appearance and Holistic View: Dual-Source Deep Neural Networks for Human Pose Estimation’, Apr. 2015.
[28] K. Toyoda, M. Kono, and J. Rekimoto, ‘Post-Data Augmentation to Improve Deep Pose Estimation of Extreme and Wild Motions’, Feb. 2019.
[29] ‘TorchVision Object Detection Finetuning Tutorial — PyTorch Tutorials 1.6.0 documentation’. https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html (accessed Oct. 21, 2020).

ANNEXE

In this annexe we will introduce some useful Python programming tools that have been used for the development of this project.

A.1 Open-source Libraries

Numpy

This library is a very commonly used library in Python and it's used for creation and calculous of multidimensional vectors that are processed by CPU or by GPU. It integrates a lot of helper functions like random generators, Fourier Transform functions and more.

PyTorch

Pytorch is an open-source library from Facebook. It works with tensors, which are multidimensional arrays that are ready to be processed by GPU which increases drastically the velocity of processing. Therefore, PyTorch is commonly used in ML environments.

It also contains a lot of helper functions and classes related to ML that allow us to work with different architectures, training techniques, datasets and more. It also has some pre-trained models for Image, voice and text and a lot of tutorials that help the developer familiarize himself or herself with the library.

TensorFlow

TensorFlow is also an open-source ML library that works with tensors. As it has been developed by Google, it is the direct competitor of PyTorch. It also contains functions that let us create, train and test our own datasets. There is a set of functions that let the user move a model from PyTorch to TensorFlow and vice versa.

Matplotlib

It's a library that allows us to represent graphs and modify them according to our needs. It has a lot of similarities with Matlab plots. Graphs used in this Master Thesis have been mostly generated with the help of this library. Some examples are Fig. 3.5, Fig. 3.9 and Fig. 4.5.

A.2 IDEs

Integrated Development Environments are software applications that allow developers to write and run codes. Some recommended IDEs are the following:

IDLE

This IDE uses the power Shell prompt. It is recommended for small applications since it provides no facilities. We can write our code in any notebook program, save it with the .py format and then execute it from the command prompt.

PyCharm

PyCharm is an example of IDE that integrates several useful tools like text editor with text helpers, code debugging, unit tester and more. A similar example like PyCharm is Spyder.

Notebook Jupyter

When we run a Jupyter Notebook a webApp is opened in our browser. From there, we can surf through our files and open notebook files (.ipynb). These contain several blocks of code that can be computed independently from the others. It is very useful when we want to run small amounts of code without computing all at once. Its calculous are performed locally; in our computer, not in the cloud and the packages depend on the ones installed in the framework environment.

Google collab

This IDE also uses Notebooks but in this case they are stored online. Their main point is that instead of running the code in our local machine, the code is executed remotely in a google machine. We can also use GPUs and TPUs for free, but with limitations. In case that we reach such limitations, we can pay to upgrade our account to Colab Pro or wait for some time (hours or a few days) until we can use their resources again. However, you can obtain Colab Pro just if you execute your codes from the United States. This function has not arrived to Europe yet.

That is to say, even if our computer is not very powerful or it has no good GPU to perform ML calculous, we can connect to this server and execute our codes remotely. Once we obtain some results, we can store them locally or in Google Drive.

A.3 Framework

Anaconda

Anaconda is a very useful tool that lets you create programming environments in which you can download and install the proper packages for your project. Different versions of the packages are available. Once the environment is set, you can run programs from any of the before mentioned IDEs or from the power shell.